UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA

Fernando Jorge Mota

Guru da Matrícula: Um simulador de matrículas multi-institucional e com suporte a análise de pré-requisitos

FLORIANÓPOLIS
2019

FERNANDO JORGE MOTA

Guru da Matrícula: Um simulador de matrículas multi-institucional e com suporte a análise de pré-requisitos

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis, dezembro de 2019

UNIVERSIDADE FEDERAL DE SANTA CATARINA

FERNANDO JORGE MOTA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciências da Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Márcio Bastos Castro
Universidade Federal de Santa Catarina - UFSC

Prof. Dr. Antônio Carlos Mariani
Universidade Federal de Santa Catarina - UFSC

Prof. Dr. Cristian Koliver
Universidade Federal de Santa Catarina - UFSC

Florianópolis, 9 de dezembro de 2019

Este trabalho é dedicado aos meus pais, Lúcia Jorge Mota e Wolnei de Matos Mota, que sempre me motivaram e me deram suporte para que eu realizasse todos os meus sonhos.

Agradecimentos

Agradeço primeiramente à minha família, que sempre me ensinou e me apoiou mesmo nos momentos mais díficeis do curso. Agradeço, também, ao professor Márcio Bastos Castro pela orientação e dedicação junto a este trabalho, assim como aos membros da banca avaliadora.

Agradeço aos professores e professoras, que, desde o ensino fundamental até o ensino superior, ensinaram os fundamentos para que eu sempre conseguisse avançar para a próxima etapa, para sempre procurar ser melhor. Agradeço também aos demais servidores e funcionários das instituições de ensino nas quais estudei, incluindo, aqui, a UFSC, sem as quais o próprio âmbito da instituição de ensino não seria tão propício ao estudo.

Agradeço também aos meus amigos e colegas, que compartilharam muitas das batalhas para conseguir chegar onde chegamos, e que, com amizade e vontade de ajudar, ajudaram a tornar o caminho mais fácil de suportar. Agradeço às pessoas que se tornaram minhas amigas durante o curso, e que não só compartilharam das dificuldades mas também proporcionaram momentos de descontração para balancear com os momentos mais díficeis do curso.

Agradeço à todos que permitiram que o conhecimento humano chegasse onde chegou, que combateram a ignorância, que verificaram e provaram suas teorias e descobertas mediante o método científico, e que hoje permitem que tenhamos o conhecimento que a humanidade possui.

Por fim, agradeço ao povo brasileiro, por apoiar e garantir o acesso aberto à educação pública e acessível, sem a qual eu jamais teria aprendido tudo o que aprendi.

Resumo

Simuladores de matrículas são ferramentas desenvolvidas para auxiliar o estudante a encontrar um quadro de horários adequado durante o período da rematrícula. Ao longo dos anos, foram desenvolvidos diversos simuladores para diferentes universidades, com diferentes conjuntos de recursos. O objetivo deste trabalho é propor um simulador de matrícula adaptado para o uso por múltiplas universidades e capaz de detectar problemas comuns relativos ao processo de rematrícula durante a simulação, como conflitos de horários e pré-requisitos de disciplinas. A motivação para tal trabalho está relacionada a dificuldade de implementação de um simulador de matrículas para outras universidades, mesmo quando os dados necessários estão disponíveis publicamente. Esse novo simulador de matrículas, determinado Guru da Matrícula, foi desenvolvido e implementado utilizando tecnologias e práticas atuais para desenvolvimento de sistemas distribuídos. Junto a esse simulador, foram então realizados diversos testes e experimentos para garantir o desempenho e baixo consumo de recursos do sistema, mesmo sob condições de grande estresse, como a importação de grandes quantidades de dados das universidades. A implementação desse sistema foi dividida em três partes: o *backend*, que implementa toda a lógica genérica do sistema, incluindo persistência de dados, gerenciamento de usuários e afins, o *frontend*, que é a interface entre o usuário e o sistema, e os extratores de dados, que são responsáveis por capturar os dados da Universidade Federal de Santa Catarina (UFSC) e da Universidade de São Paulo (USP) e transformá-los em arquivos *JavaScript Object Notation* (JSON) com uma estrutura comum, preparados para importação pelo *backend*. Em relação aos resultados observados nos testes, observou-se um desempenho considerado aceitável, com o sistema sendo capaz de executar mesmo em especificações limitadas, com apenas 512MB de *Random Access Memory* (RAM), e com a possibilidade de configurar o sitema para usar mais máquinas, de maneira distribuída, o que permite escalar facilmente para ainda mais universidades e usuários.

Palavras-chave: simulador, matrículas, *web*, estudantes, planejamento.

Abstract

Enrollment simulators are tools designed to help students to find an appropriate time-sheet during the re-enrollment period. Over the years, various simulators have been developed for different universities with different feature sets. The aim of this work is to propose an enrollment simulator adapted for use by multiple universities and capable of detecting common problems related to the re-enrollment process during the simulation, such as time conflicts and subject prerequisites. The motivation for such work is related to the difficulty of implementing an enrollment simulator for other universities, even when the necessary data is already publicly available. This new enrollment simulator, named Guru da Matrícula, was developed and implemented using current technologies and practices for distributed system development. Along with this simulator, several tests and experiments were then performed to ensure the performance and low consumption of system resources, even under conditions of high stress, such as the importation of large amounts of data from universities. The implementation of this system has been divided into three parts: the backend, which implements all generic system logic, including data persistence, user management and the like, the frontend, which is the interface between the user and the system, and the data extractors, who are responsible for capturing UFSC and USP data and turning it into common structure JSON files prepared for the *backend* import into database. Regarding the results observed in the tests, it was considered an acceptable performance, with the system being able to perform even in limited specifications, with only 512MB RAM, and with the possibility of configuring the system to use more machines, in a distributed way, allowing you to easily scale to even more universities and users.

Keywords: simulator, enrollment, *web*, students, planning.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Os alunos dos cursos de graduação têm diversas dificuldades no momento de realizar a rematrícula. Uma delas é encontrar quais disciplinas eles podem cursar - dado o cumprimento dos pré-requisitos conforme imposto pela universidade em que estuda. Esses pré-requisitos comumente são disponibilizados em uma página ou em um arquivo contendo todo o currículo do curso, juntamente com outros dados das disciplinas, como disciplinas equivalentes, carga horária e ementa.

Entretanto, os sistemas acadêmicos das universidades são, muitas vezes, pouco intuitivos ou apresentam restrições nas informações fornecidas ao aluno. Isso acontece ao ponto deles não realizarem validações básicas tais como apresentar um erro mostrando que o aluno não pode se matricular em uma disciplina por não ter cumprido todos os seus pré-requisitos ou mostrar erro frente a outros problemas, como conflitos de horários entre turmas distintas. Esses fatores dificultam ainda mais o processo de escolha e de decisão dos alunos.

Durante o período de rematrícula, para tomar suas decisões, os alunos não levam somente em consideração os pré-requisitos, mas também outros aspectos, como as escolhas dos colegas mais próximos, atividades extra-curriculares que o aluno pretende fazer no período letivo seguinte, e também combinações que não tenham conflito de horários entre si - pois não são um plano válido em qualquer universidade.

Dado a grande quantidade de disciplinas que os cursos possuem e os fatores limitantes anteriormente mencionados, realizar a matrícula não é uma tarefa simples na grande maioria dos casos. Isso faz com que os estudantes, por vezes, demorem até escolher algum plano de turmas definitivo para o próximo período. Além de fazer com que o plano proposto pelo aluno para a rematrícula seja suscetível a problemas que aparecerão posteriormente, na confirmação definitiva da matrícula, que é liberada após alguns dias pela própria universidade.

Para resolver esse problema, foram propostas ferramentas com o objetivo de auxiliar os alunos a escolher as turmas no qual desejam se matricular, conhecidas como **simuladores de matrículas**. Essas ferramentas recebem dados da universidade, seja de modo manual ou automático, e então disponibilizam para os alunos uma interface mais intuitiva, além de recursos que facilitam a busca pela melhor combinação de disciplinas e turmas.

O primeiro simulador de matrículas, denominado Grade de Matrícula (GRAMA), foi desenvolvido nos anos 2000 por um estudante de engenharia da produção da UFSC chamado Glauco Peron e orientado pelo professor Sérgio Mayerle. Essa ferramenta provocou

um impacto tão grande no processo de matrícula da universidade que chegou a receber apoio oficial da mesma pouco tempo após sua criação.

Posteriormente, diversas outras ferramentas do tipo surgiram não só na UFSC como também em outas universidades. Alguns exemplos são o **Meu Horário**, que atende a Universidade Federal da Bahia (UFBA); o **GDE**, criado para atender a Universidade de Campinas (UNICAMP); o **MatrUFSC**, alternativa ao GRAMA desenvolvida também para os alunos da UFSC; o **MatrUSP**, sistema baseado no MatrUFSC mas adaptado para atender os alunos da USP; o **Grade na Hora!**, o qual foi desenvolvido para atender os estudantes da Universidade Tecnológica Federal do Paraná (UTFPR); o **MatrUFSC2**, alternativa ao MatrUFSC desenvolvido com o objetivo de fornecer mais recursos aos estudantes da UFSC; e, por fim, o **Meu Horário 2**, que atende a UFBA e é uma evolução do **Meu Horário**.

1.1 Motivação

Infelizmente, as ferramentas mencionadas anteriormente não resolvem completamente o problema que os alunos têm na hora de fazer a rematrícula: encontrar quais disciplinas o aluno **pode** realmente fazer. Atualmente, a maior parte das ferramentas considera apenas um ponto, bem específico, nesse aspecto: encontrar turmas cujos horários não tenham conflito entre si.

As universidades possuem hoje mecanismos complexos para decidir se o aluno pode ou não realizar a matrícula na disciplina. Um desses mecanismos é o sistema de pré-requisitos, que define um grafo direcionado entre as disciplinas que os alunos devem respeitar para poder cursar as disciplinas na ordem correta, de forma que não ocorra surpresas como o aluno não saber algo necessário para cursar a disciplina em questão, por exemplo. De todos os simuladores citados, somente o **Meu Horário 2** e o **GDE** usam esses dados.

Nesse mesmo sentido, as universidades costumam liberar, também, uma lista de disciplinas equivalentes, que correspondem às disciplinas que, se o aluno cursar, correspondem à disciplina em questão. Na maior parte das vezes, essas listas servem como alternativas às disciplinas presentes no currículo do curso, e permitem aumentar o número de possibilidades que o aluno tem a disposição na hora de escolher os horários para o semestre seguinte.

Além disso, a maior parte das ferramentas atuais implementam apenas suporte básico para atividades extra-curriculares, permitindo o cadastro básico desses dados mas sem apoiar atividades que demandam uma quantidade de horas fixa na semana, como no caso de estágios ou bolsas de iniciação científica, por exemplo.

Outro aspecto importante é o fato de que nenhum dos simuladores citados atendem mais de uma universidade. Isso é um aspecto importante pois as universidades compartilham muitos aspectos similares entre si - seja em questão de entidades envolvidas (uma vez que todas as universidades possuem turmas, disciplinas e professores, por exemplo), seja em questão de atributos envolvidos (todas as turmas possuem um horário inicial e um horário final para as aulas, por exemplo) - e ainda assim não existe hoje um sistema capaz de atender diferentes universidades baseado nesses aspectos comuns, que permitiria, entre outras coisas, a inclusão de mais universidades facilmente.

Considerando tudo isso, é simples notar que não existe hoje um sistema com suporte não só para mais de uma universidade como também com suporte à detecção de pré-requisitos entre disciplinas e outros recursos úteis voltados a facilitar a busca por horários atraentes para os estudantes, ou seja, que respeitem não só suas metas na vida acadêmica como também permitam que o estudante encontre horários adequados para estágios e demais atividades extra-curriculares.

1.2 Objetivo

O objetivo básico e fundamental deste trabalho é desenvolver um sistema que seja capaz de trabalhar com múltiplas universidades e também ajude a resolver os diferentes problemas que os estudantes possuem, que hoje consiste em encontrar horários:

- Que estejam dentro do limite de horas-aula que o curso possui;
- Cujas disciplinas envolvidas tenham todos os seus pré-requisitos já cumpridos pelo aluno;
- Que possibilitem a realização de atividades extra-curriculares, como estágios e cursos;
- Que permitam a realização das disciplinas em turmas nos quais os amigos do aluno também estão presentes, visando facilitar os estudos.

1.2.1 Objetivo Geral

O objetivo geral é desenvolver um sistema que seja capaz de trabalhar com múltiplas universidades, implementando suporte primário aos pré-requisitos das disciplinas - ou seja, realizando desde validação até sugestão de disciplinas para o aluno com base no seu curso e histórico de disciplinas feitas - e também a disciplinas equivalentes.

Observe que pelo fato de que nem todas as universidades disponibilizarem formas de identificar disciplinas que são pré-requisitos ou equivalentes, o sistema deverá permitir, ainda assim, o cadastro de ofertas de turmas e horários facilmente, mas, nesse caso em

específico, sem disponibilizar recursos contextuais como análise de pré-requisitos, por falta da informação.

Dado esses pontos, a ideia por trás do presente trabalho é de propor um sistema que possibilite cadastrar uma nova universidade no sistema com base no conceito de *web scraper*[1] ou outro mecanismo capaz de exportar os dados de turmas e disciplinas que a universidade possui em um formato comum previamente especificado. Depois, um simples cadastro no sistema deverá bastar para o sistema rastrear os dados que a universidade possui e disponibilizá-los para os alunos, que poderão acessá-los através de computadores ou dispositivos móveis e terão a possibilidade de - além de encontrar horários para as turmas - também encontrar horários adequados para atividades extra-curriculares, e ainda poder salvar os planos criados para acesso posterior.

1.2.2 Objetivos Específicos

Considerando o desenvolvimento do trabalho e o objetivo geral apresentado, destacam-se os seguintes objetivos específicos:

- Desenvolvimento de um mecanismo de captura de dados para UFSC e USP, com saída comum dos dados;
- Definição de um modelo comum para transmissão e armazenamento de dados de turmas de diferentes universidades;
- Desenvolvimento de um sistema para detecção e reconhecimento de disciplinas equivalentes e pré-requisitos para auxílio na simulação da matrícula;
- Desenvolvimento da ferramenta *web* para criação e salvamento de planos;
- Desenvolvimento de opção para gerenciamento de atividades extra-curriculares.

1.3 Organização do Documento

O presente trabalho está organizado da seguinte forma:

Capítulo 2 Apresenta o contexto por trás do desenvolvimento dos simuladores de matrículas existentes no Brasil, com uma breve apresentação a respeito da história dos mesmos, seus respectivos problemas e também o impacto que tiveram entre os estudantes das universidades que atendem.

[1] Ferramenta ou técnica criada com o objetivo de realizar extração de dados de uma página *web* em um formato estruturado.

Capítulo 3 Apresenta a proposta do presente trabalho, discutindo o método através do qual foram definidos os requisitos para o projeto e uma visão geral do que foi feito.

Capítulo 4 Discute as escolhas tomadas para o desenvolvimento do protótipo do sistema e as razões por trás destas escolhas, incluindo decisões relacionadas à captura e ao armazenamento dos dados.

Capítulo 5 Apresenta o estudo de caso a partir da implementação do suporte à duas universidades, UFSC e USP, com a apresentação do método usado para a captura dos dados e detalhes da abordagem usada;

Capítulo 6 Discute os resultados a partir da implementação desenvolvida para o sistema, incluindo análise dos mecanismos de coleta de dados das universidades e comentários gerais sobre os dados coletados;

Capitulo 7 Apresenta o cronograma do projeto e também uma conclusão em relação ao contexto dos simuladores de matrícula com a proposta de desenvolvimento e as escolhas tomadas para o presente trabalho.

2 Os Simuladores de Matrícula

Os simuladores de matrícula surgiram principalmente devido à falta de determinados recursos no sistema das universidades e também pelo fato de que, normalmente, é demorado encontrar um conjunto de horários que pareça adequado para o estudante. Até a presente data, nenhum simulador de matrícula possui, entre seus recursos principais a possibilidade de ser genérico, ou seja, de adicionar, com pouca ou nenhuma mudança, suporte a mais universidades de forma simples.

As seções a seguir apresentam alguns detalhes sobre os simuladores existentes propostos no âmbito de diferentes universidades brasileiras.

2.1 Universidade Federal de Santa Catarina (UFSC)

A UFSC é a universidade que durante sua história possuiu mais simuladores de matrículas em comparação com as outras. Isso se deve ao fato de que a universidade publica uma lista de turmas todo semestre algum tempo antes do período de matrículas e, a cada período de matrículas, é necessário aguardar um período (que normalmente é de uma semana) até o final do processamento da rematrícula de todos os estudantes para poder, então, ver qual o resultado do processo. Além disso, não é possível consultar procurar, no sistema de rematrículas, por combinações válidas de horários entre um determinado conjunto de turmas e disciplinas. Graças a isso, os simuladores de matrícula se tornaram uma ferramenta popular entre os estudantes, que procuram usar os simuladores antes de cadastrar o plano no sistema de graduação da universidade.

O primeiro simulador de matrículas que se tem notícia foi desenvolvido para a UFSC e se chama GRAMA. Ele foi criado no início dos anos 2000, desenvolvido por um estudante de Engenharia de Produção da própria universidade. Esse simulador foi o único que chegou a receber apoio oficial da UFSC, chegando inclusive a ser hospedado em um subdomínio cedido pela própria universidade. No entanto, em 2012, devido a problemas envolvendo o GRAMA, a UFSC removeu o seu apoio oficial ao site e assim este se tornou indisponível para acesso pelos estudantes.

No mesmo ano, foi disponibilizado uma alternativa - que começou a ser desenvolvida por um aluno da Engenharia Elétrica já em 2011 sob o nome Combinador Automático de Possibilidades Interativo de Matrícula (CAPIM) e publicado sob o nome **MatrUFSC**[1]. Dentre os novos recursos adicionados, estava a comunicação feita através de *Asynchronous Javascript And XML* (AJAX), que permitia que o carregamento das disciplinas fosse

[1] <http://pet.inf.ufsc.br/matrufsc/>

realizado de maneira assíncrona, e uma nova licença visando garantir que o MatrUFSC não viesse a ter os mesmos problemas do GRAMA no futuro.

Em abril de 2014, o criador do MatrUFSC contatou o grupo do curso de Ciências da Computação da UFSC no Facebook procurando por pessoas interessadas em manter o MatrUFSC, visto que em 2013 ele havia se formado e não tinha mais como manter o site. Diante disso, o MatrUFSC original foi transferido para o PET Computação e foi iniciado o desenvolvimento do **MatrUFSC2**[2] pelo autor do presente trabalho, sendo este liberado já no segundo semestre de 2014 para os alunos, com uma nova interface, código totalmente novo e infra-estrutura distribuída.

2.1.1 GRAMA

O **GRAMA** foi desenvolvido no inicio dos anos 2000 pelo aluno Glauco Peron e orientado pelo professor Sérgio Mayerle com o objetivo de permitir que os alunos pudessem planejar seus horários de matrícula.

Dentre seus pontos positivos estão o fato de possuir integração total com o banco de dados da UFSC, a ponto de não ter atrasos em relação à atualização das disciplinas, e a possibilidade de salvar planos, incentivando os usuários a identificarem o plano com o número de matrícula do aluno na UFSC.

Entretanto, sua interface era muito básica e exigia a digitação dos códigos das disciplinas, além de não conseguir verificar pré-requisitos ou disciplinas equivalentes, mesmo com o acesso ao banco de dados interno da UFSC. Além disso, o sistema não era genérico - justamente pela integração forte com o banco de dados da universidade - e não podia portanto ser facilmente integrado com outras universidades.

Por fim, o sistema não possuía código aberto. Logo, quando seu criador resolveu anunciar a empresa que havia recém-criado no site e a UFSC removeu seu apoio ao sistema, o site foi simplesmente tirado do ar em 2012 e os alunos acabaram simplesmente migrando para o MatrUFSC, que foi disponibilizado no ano seguinte. Hoje, pouquíssimas informações a respeito do GRAMA e suas funcionalidades são encontradas *online*.

2.1.2 MatrUFSC

O **MatrUFSC** foi desenvolvido em 2012 por um aluno da Engenharia Elétrica da UFSC chamado Ramiro Polla focando em criar uma alternativa de código aberto para o GRAMA que, pela licença, não permitisse que eventuais anúncios fossem adicionados na página no futuro, tal como aconteceu com o próprio GRAMA. A Figura 1 apresenta a interface principal do MatrUFSC.

[2] <https://matrufsc2.appspot.com>

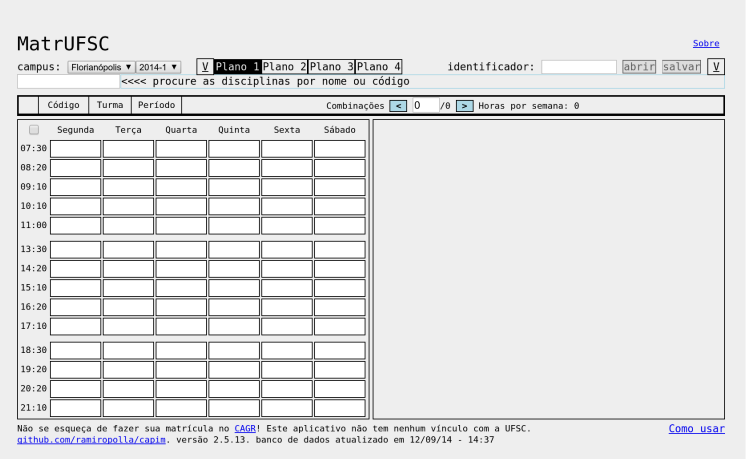

Figura 1 – Página inicial do **MatrUFSC**.

Dentre suas funcionalidades, estão um mecanismo de combinações com detecção de conflitos de horários, acesso dos dados assincronamente via AJAX e funcionamento baseado em *Single Page Application* (SPA) (ou seja, todo o processo de simulação de matrícula é feito através de uma única página).

Além disso, estão entre seus principais recursos a busca por nome da disciplina ou código, o salvamento de planos à partir de um identificador, e também a possibilidade do aluno definir múltiplos planos, tal como acontece no CAGR[3]. Outra característica importante é que o **MatrUFSC** possui código aberto, disponível sob o nome CAPIM.

Entre os pontos negativos, entretanto, estão o fato de que os identificadores podem ser sobrescritos por outros usuários, pois não há controle ou autenticação, além de não haver suporte a outras universidades de maneira genérica, sendo necessário adaptar o sistema em termos de código para poder lidar com os dados de outras universidades. Por fim, também não há suporte para validação de pré-requisitos e sugestão de disciplinas equivalentes ou filtragem de outros tipos que não o nome ou o código da disciplina.

2.1.3 MatrUFSC2

O **MatrUFSC2** foi desenvolvido em 2014 pelo autor do presente trabalho como alternativa mais avançada e moderna em relação ao **MatrUFSC**, mas ainda com código aberto e seguindo os mesmos princípios contra eventuais anúncios ou mal uso da ferramenta. A Figura 2 apresenta a interface principal do MatrUFSC2.

Dentre suas funcionalidades principais, pode-se destacar a capacidade de sugerir correções na busca para termos possivelmente errados (como erros de digitação, por exemplo), versionamento para cada plano criado, planos públicos, uma central de ajuda com conteúdo sobre o processo de rematrículas da UFSC e também opções capazes de filtrar

[3] O CAGR é o sistema acadêmico da UFSC.

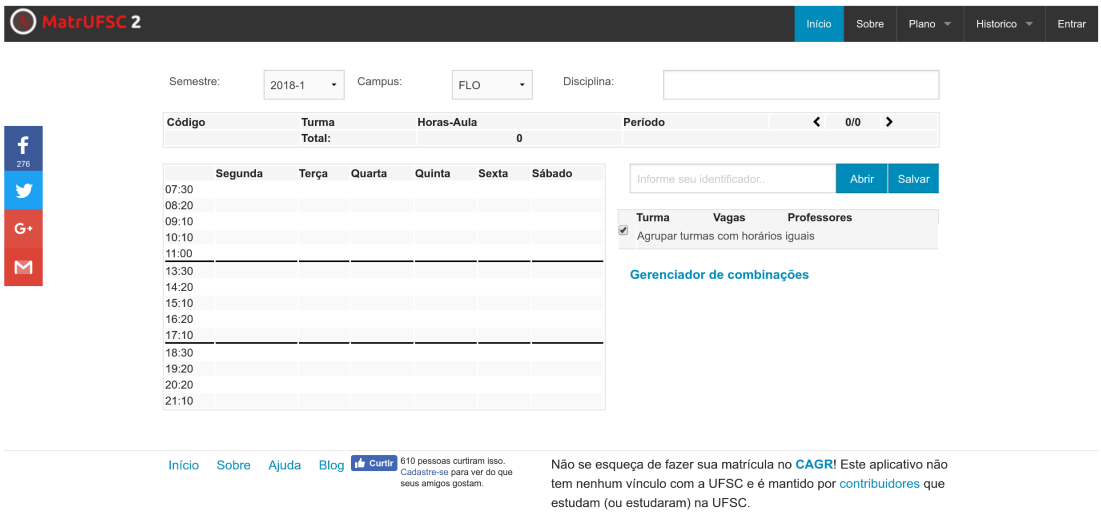

Figura 2 – Página inicial do **MatrUFSC2**.

as combinações processadas pelo sistema por dia da semana e também por quantidade de horas-aula. Por fim, o mecanismo também conta com sistema de autenticação via Google, o qual evita que um mesmo identificador seja escrito por mais um usuário inadvertidamente. A lista dos campi é ordenada automaticamente utilizando-se a distância entre a localização geográfica dos mesmos e o IP do usuário que está usando o sistema.

Entretanto, o sistema também tem variados pontos negativos, como falta de suporte para validação de pré-requisitos e suporte a disciplinas equivalentes. Além disso, o MatrUFSC2 também possui alguns problemas relacionados a busca - que esporadicamente acaba por trocar os resultados dos campi e apagar disciplinas indevidamente devido a problemas na arquitetura distribuída do sistema, embora normalmente tais informações sejam restauradas após algumas horas de maneira automática.

2.2 Universidade Federal da Bahia (UFBA)

A UFBA possui mais ou menos os mesmos problemas que a UFSC durante o período da rematrícula: O sistema não mostra as diferentes combinações de horários entre turmas e não mostra erro imediato mediante problemas na matrícula. Além disso, a universidade publica, assim como a UFSC, os dados sobre os horários da turmas e suas respectivas disciplinas de modo público, no SUPAC[4], permitindo portanto que os simuladores capturem os dados que precisam para poder disponibilizar aos estudantes.

O primeiro simulador de matrículas adaptado para trabalhar com os dados da

[4] O SUPAC é o sistema acadêmico da UFBA.

Figura 3 – Página inicial do **Meu Horário**.

UFBA surgiu em 2004 e se chama **Meu Horário**[5]. O **Meu Horário** foi desenvolvido por Rodrigo Rocha e, diferente do GRAMA e do CAPIM, trabalha com as disciplinas individualmente por curso, mas sem lidar, ainda, com pré-requisitos ou outros aspectos específicos ao curso e a universidade, mas permitindo a impressão dos planos feitos, o cadastro de disciplinas optativas e a filtragem por horários e por professores.

Em 2017, foi desenvolvido - como evolução ao **Meu Horário** - o **Meu Horário 2**[6], sob a forma de um TCC [Erbetta 2017] da própria UFBA feito pelo aluno Gabriel Erbetta e orientado pelo criador do **Meu Horário**. O sistema apresentou diversas melhorias, tais como uma interface adaptada para dispositivos móveis e a possibilidade de lidar com os dados de pré-requisitos das disciplinas fornecidos pela UFBA, embora não possua código aberto disponibilizado, o que impede que a comunidade ajude com a manutenção e melhorias para o site.

2.2.1 Meu Horário

O **Meu Horário** foi desenvolvido em 2004 pelo aluno Rodrigo Rocha para uso pelos alunos da UFBA. Dentre suas características principais, estão uma separação estrita das disciplinas por curso e o suporte a associar um apelido a uma disciplina, recursos incomuns em relação aos outros simuladores de matricula. A Figura 3 apresenta a interface principal do Meu Horário.

Outros recursos interessantes que o Meu Horário possui comparado com outros sistemas é o suporte a definir restrições de horário (que funciona de maneira parecida

[5] <http://meuhorario.dcc.ufba.br/index2.php>
[6] <http://meuhorario.dcc.ufba.br>

com a definição de atividades extra-curriculares em outros mecanismos, mas que aqui é demonstrado de maneira fixa) e também restrições de professores - que permitem definir os professores com quem o aluno não quer ter aula, por exemplo.

Além disso, o sistema implementa suporte explícito para disciplinas optativas, normalmente provenientes de outros cursos, e fornece suporte a versão para impressão das combinações geradas.

Entretanto, o Meu Horário também possui alguns pontos negativos, entre eles, estão a falta de suporte a detecção de pré-requisitos e a impossibilidade de salvar planos diretamente no servidor para acesso posterior.

2.2.2 Meu Horário 2

O **Meu Horário 2** foi desenvolvido em 2017 por Gabriel Assis Erbetta com o objetivo de criar uma alternativa ao Meu Horário com mais recursos e plataforma mais moderna, enquanto que revisando aspectos como design e usabilidade para uso por celulares. A Figura 4 apresenta a página principal do Meu Horário 2.

Esse sistema foi desenvolvido como um trabalho de TCC [Erbetta 2017] na própria UFBA e orientado pelo próprio criador do Meu Horário. Frente a esse, possui como recursos novos a exibição da lista de pré-requisitos de cada disciplina (com exibição da grade de pré-requisitos do curso, inclusive), melhor interface para dispositivos móveis e busca de disciplinas, enquanto mantem algumas características do seu precursor, como por exemplo acesso de disciplinas por curso, em vez de por campus, como acontece em outros simuladores.

Entretanto, há alguns pontos negativos: não é possível salvar planos no servidor - problema este que já existia no Meu Horário - e também não é possível restringir horários e professores tal como seu precursor fazia, o que impede que os estudantes cadastrem eventuais horários do estágio e de demais atividades extra-curriculares, por exemplo.

Por fim, é notório o fato de que, diferente do Meu Horário, o Meu Horário 2 não possui código aberto, o que faz com que problemas e possíveis melhorias possam ser feitas apenas por seu criador e por quem tiver acesso ao código, e não pela comunidade.

2.3 Universidade de Campinas (UNICAMP)

Na UNICAMP, a universidade libera os dados de turmas e horários através de um "caderno de horários" no DAC[7] e disponibiliza a opção de rematrícula através do mesmo sistema. Entretanto, não há documentação clara sobre o processo de rematrícula, sendo

[7] DAC é o portal da diretoria acadêmica da UNICAMP.

Figura 4 – Página inicial do **Meu Horário 2**.

possível observar apenas que há um intervalo até a confirmação da rematrícula (assim como na UFSC).

Sendo assim, foi desenvolvido entre 2009 e 2012 pelo aluno de Ciências da Computação Felipe Guaycuru uma rede de auxílio acadêmico chamada **GDE**[8], que, entre outras coisas, possui um simulador de matrículas.

Em 2016, o **GDE** começou a ser reescrito com o foco de otimizar o sistema principalmente em questão de segurança. Além disso, foram feitas outras melhorias variadas como, por exemplo, a implementação do suporte a oferecimentos de turmas com mais de um professor e também suporte a múltiplas disciplinas com a mesma sigla. Entretanto, essa versão do sistema ainda não foi liberada para os usuários, uma vez que ainda está em desenvolvimento.

2.3.1 GDE

O **GDE** foi desenvolvido pelo aluno de Ciências da Computação Felipe Guaycuru entre 2009 e 2012 com o propósito de criar uma rede integrada com diversos recursos para a UNICAMP. Diferentemente dos outros projetos abordados até então, o GDE possui um conjunto de recursos que trabalham não só no sentido de permitir ao usuário simular sua matrícula, mas também disponibilizar outras informações como mapa da universidade, cardápio do restaurante universitário e uma rede social para comunicação entre os estudantes. A Figura 5 apresenta a página principal do GDE.

[8] <https://grade.daconline.unicamp.br/>

Figura 5 – Página inicial do **GDE**.

O principal ponto positivo do GDE em relação às outras alternativas é que, por contar com uma grande quantidade de recursos integrados diretamente à universidade, é possível que o simulador se aproveite dos demais dados para sugerir assim os melhores horários. Dessa forma, o sistema acaba por disponibilizar a árvore de pré-requisitos para acesso rápido e também fornecer formas do aluno obter as informações que precisa para encontrar o melhor horário - com o fornecimento do mapa do campus, por exemplo.

A versão atualmente em produção não possui código aberto. Entretanto, uma nova versão do sistema foi desenvolvida em 2016. Esta nova versão possui mais recursos, tais como suporte a turmas com mais de um professor e ajustes para permitir turmas com a mesma sigla, além de ser mais focado em segurança e possuir código aberto. Entretanto, ainda não é a versão usada atualmente em produção. Além disso, é importante notar que, embora todo o sistema seja hospedado dentro da própria universidade, o GDE não possui apoio oficial da UNICAMP.

2.4 Universidade de São Paulo (USP)

No caso da USP, o Júpiter Web[9] disponibiliza os dados de horários e turmas para todas as disciplinas e campus da universidade. Além disso, o sistema da USP mostra apenas disciplinas cujos pré-requisitos já foram cumpridos, algo que o diferencia do sistema acadêmico da UFSC, por exemplo, que não faz esse tipo de filtragem ao mostrar a lista de disciplinas que o aluno pode escolher para o período letivo seguinte.

O primeiro simulador de matrículas criado para atender os alunos da USP foi o **MatrUSP**[10]. Ele foi desenvolvido por Pedro Paula Vezzá Campos em 2013 a partir de adaptações no código do MatrUFSC, que Pedro conhecera enquanto estudava na UFSC, logo antes de ser transferido para a USP. Dentre as modificações feitas para adaptar o MatrUFSC à realidade da USP estão o desenvolvimento de um novo *web scraper* para

[9] Sistema acadêmico da USP.
[10] <http://bcc.ime.usp.br/matrusp_v1/>

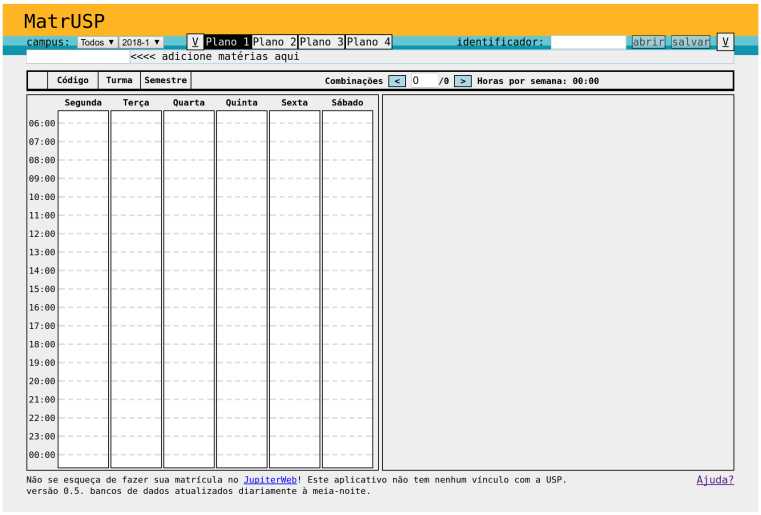

Figura 6 – Página inicial do **MatrUSP**.

captura dos dados a partir do sistema da USP e também a adaptação da interface para lidar com a grade de horários diferenciada.

Ainda em 2016 foi desenvolvido um TCC [Endo, Rodrigues e Verona 2016] na própria USP visando desenvolver uma nova versão do **MatrUSP**[11], especialmente no aspecto de usabilidade pelos usuários. Esse trabalho foi desenvolvido por Bruno de Oliveira Endo, José Ernesto Young e Rafael Marinaro Verona sob orientação do desenvolvedor original do MatrUSP e do Prof. José Coelho de Pina. Dentre os seus principais diferenciais pode-se destacar a sua nova interface, baseada em princípios específicos de *design* e a pré-visualização das combinações de horários diretamente através de miniaturas.

2.4.1 MatrUSP

O **MatrUSP** foi desenvolvido em 2013 por Pedro Paula Vezzá Campos a partir de adaptações do código do MatrUFSC visando integrá-lo com o modelo de dados que a USP utiliza, o qual é diferente do modelo de dados da UFSC, e assim disponibilizá-lo para a universidade paulista. A Figura 6 apresenta a página principal do MatrUSP.

O sistema possui as mesmas características que o MatrUFSC em termos de recursos, tais como: suporte a múltiplos planos, busca de disciplinas por nome ou código, salvamento de planos, código aberto, ausência de suporte a validação de pré-requisitos ou disciplinas equivalentes. Portanto, ele não possui nenhum recurso novo em comparação com o MatrUFSC. Todavia, ele se destaca principalmente pela interface mais adaptada ao modelo de dados da USP, que não possui uma grade de horários muito bem definida e também não fornece dados sobre semestres anteriores ou de múltiplos campus (no caso, a USP não possui esses filtros de dados), tal como a UFSC fornece.

[11] <http://bcc.ime.usp.br/matrusp/>

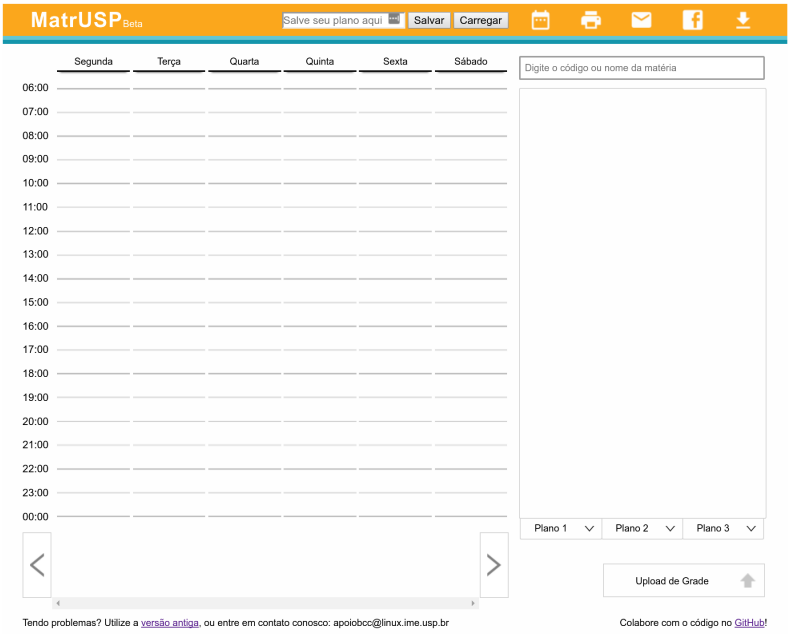

Figura 7 – Página inicial da **nova versão do MatrUSP**.

2.4.2 MatrUSP (Nova Versão)

A **nova versão do MatrUSP** foi desenvolvida como um TCC [Endo, Rodrigues e Verona 2016] em 2016 pelos alunos da USP Bruno de Oliveira Endo, José Ernesto Young e Rafael Marinaro Verona e orientado pelo criador do MatrUSP, Pedro Paula Vezzá Campos. Esse projeto foi desenvolvido com o intuito principal de revisar completamente o *design* da ferramenta, visando torná-lo mais amigável para os usuários. A Figura 7 apresenta a página principal da nova versão do MatrUSP.

A nova versão do MatrUSP conta com basicamente os mesmos recursos do MatrUSP: busca de disciplinas por nome ou código, armazenamento de identificadores no servidor, uso de AJAX para realização das requisições ao servidor e arquitetura SPA para realização das simulações em uma única página. Porém, ele apresenta algumas melhorias como um *design* totalmente renovado e baseado em conceitos de Interação Humano-Computador (ICH), com um novo painel para visualização das combinações disponíveis de turmas de maneira visual. Além disso, possui código aberto e utiliza-se do mesmo *web scraper* responsável pela indexação de dados para o MatrUSP original.

Em termos de design, a maior diferença observada é a ausência de alguns campos como semestre e campus, que no MatrUSP original só disponibilizavam uma única opção e portanto não eram úteis em termos de design. Como pontos negativos, tal como o MatrUFSC e o MatrUSP, ele não possui nenhuma forma de validação para pré-requisitos ou disciplinas equivalentes e também não possui nenhuma proteção contra sobrescrita dos identificadores de planos.

2.5 Universidade Tecnológica Federal do Paraná (UTFPR)

Na UTFPR, não há disponibilização pública dos dados de turmas e horários, sendo necessário portanto dados de acesso do sistema da universidade como aluno para poder acessar esses dados. Esse sistema é capaz de sugerir um quadro de horários baseado nas disciplinas obrigatórias que o aluno ainda precisa fazer, e é simples visualizar as disciplinas do curso que o aluno ainda não cursou. Entretanto, não é possível saber, assim como no caso da UNICAMP, se o sistema é capaz de validar os pré-requisitos das disciplinas de forma imediata antes de enviar o requerimento para a universidade.

Nesta universidade, o primeiro simulador de matrículas desenvolvido com o propósito de atender essa universidade foi o **Grade na Hora!**[12], que foi desenvolvido em 2012 por alunos do curso de Engenharia da Computação. Dentre seus recursos, estão a possibilidade de definir um limite de créditos na grade - tal como o **MatrUFSC2** permite - e contar com uma visualização das disciplinas por curso - tal como o **Meu Horário** faz.

A versão inicial desse sistema exigia que fosse copiado, na própria aplicação, o código da página de turmas disponíveis para o curso desejado pelo aluno. No mesmo ano do lançamento, entretanto, foram feitas melhorias e alguns cursos foram pré-cadastrados no mecanismo, eliminando assim a necessidade de acesso ao sistema acadêmico pelos usuários para a obtenção dos dados. A Figura 8 apresenta a interface principal do Grade na Hora!, com parte da lista de campus mostrada (a página inicial desse sistema mostra todos os campus da universidade, mas na figura tal lista foi omitida por brevidade).

Entretanto, o sistema não oferece salvamento dos planos no servidor, assim como também não há suporte para validação de pré-requisitos nem suporte a um sistema de combinações tal como acontece nos outros simuladores - ou seja, cada turma é selecionada manualmente, e não cada disciplina, e caso uma disciplina tenha mais de uma turma selecionada ambas aparecem no painel, deixando ao usuário a tarefa de resolver conflitos e verificar combinações. Além disso, ele não conta com atualização automática via *web scraper*, necessitando, portanto, de atualização manual dos horários no sistema por colaboradores.

2.6 O Impacto nas Matrículas

Mesmo com todos os seus problemas, algo inegável é o fato de que cada simulador de matrícula se tornou razoavelmente usado pelos universitários pouco tempo após seu lançamento, mostrando que os estudantes realmente gostam da ideia desses simuladores.

Na UFSC, por exemplo, temos hoje o **MatrUFSC**, que já chegou a conseguir quase 3 mil visitas no período de uma semana - e possui mais de 5 mil curtidas no Facebook - e o **MatrUFSC2**, que possui mais de 500 curtidas e tem armazenados em seus servidores

[12] <https://gradenahora.com.br>

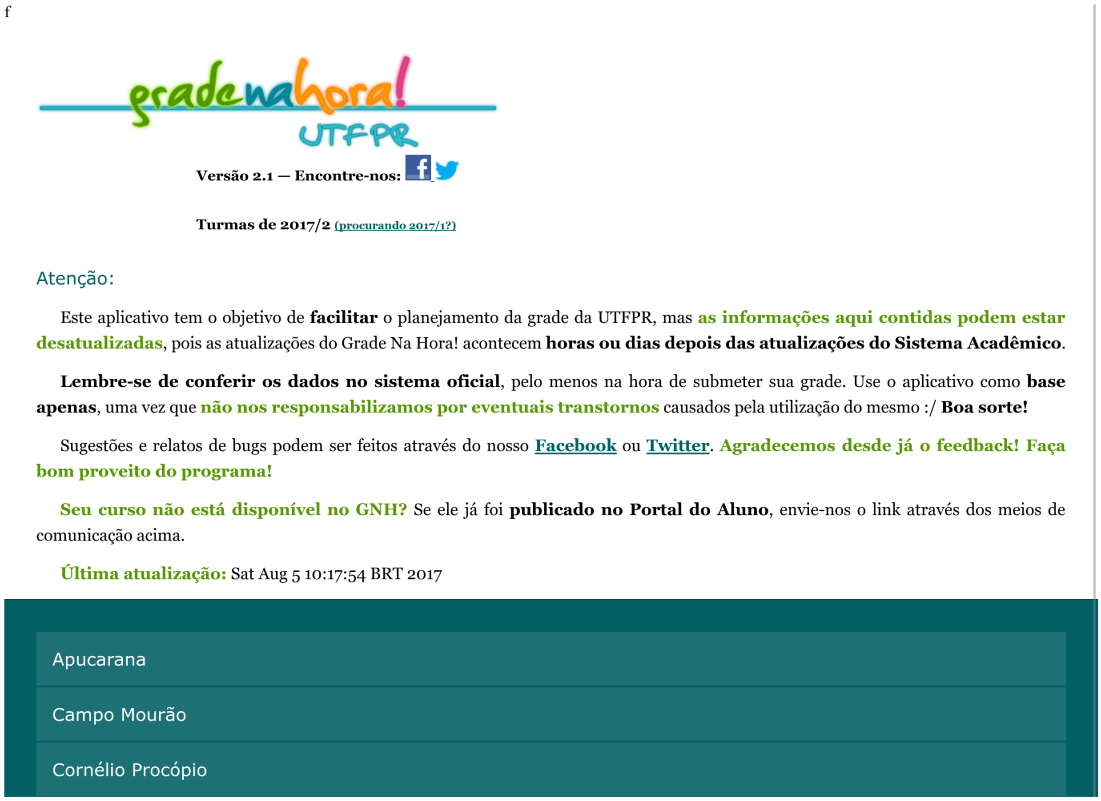

Figura 8 – Página inicial do **Grade na Hora!**.

mais de 4.700 planos, com mais de 100 mil buscas sendo feitas durante os períodos de rematrícula da universidade.

Enquanto isso, na UNICAMP, o **GDE** afirma possuir mais de 10 mil usuários, enquanto possui mais de 100 curtidas no Facebook. Entretanto, a página do Facebook não é divulgada pela página do GDE. Portanto, é esperado que esse número seja baixo em relação ao número de usuários cadastrados.

Já na USP, o **MatrUSP** possui, no Facebook, mais de 4.700 curtidas, e conseguiu cerca de 5 mil visitantes somente no período das rematrículas de 2013 - segundo informações disponíveis na própria página da plataforma no Facebook. Por outro lado, o **Grade na Hora!**, que atende a UTFPR, possui mais de 2.300 curtidas no Facebook, embora não se tenha uma estimativa aproximada do seu número de acessos divulgada.

Com isso, é possível concluir que esses simuladores tiveram - em suas respectivas universidades - impacto significativo em relação ao número total de alunos, mostrando como essas ferramentas se tornam importantes durante o período de rematrícula para os alunos.

2.7 Levantamento dos Principais Problemas dos Simuladores Analisados

Os simuladores citados anteriormente - embora tenham grande aceitação pelos alunos - ainda não resolvem muitos problemas que os alunos também enfrentam por falta de alguns recursos básicos relacionados principalmente sobre como as universidades validam a rematrícula e também o que os alunos costumam fazer durante o semestre - incluindo estágio e atividades extra-curriculares.

Dentre os problemas encontrados nos simuladores existentes, o principal é a falta de validação de pré-requisitos de disciplinas ou de condições que o aluno precisa cursar para poder realizar uma disciplina, além da falta de sugestões de disciplinas baseadas no que o aluno já cumpriu. Isto dificulta a idealização de um plano inicial visto que é necessário que o aluno consulte quais disciplinas ele pode fazer, sendo necessário checar quais disciplinas do curso já estão com os pré-requisitos concluídos. Dentre os simuladores analisados, apenas o **Meu Horário 2** e o **GDE** mostram a grade de disciplinas já com os pré-requisitos e a lista de disciplinas que será liberada. Todavia, o Meu Horário 2 não possui nenhuma forma de salvar dados no sistema, e portanto acaba por não efetuar validação nem sugestão de disciplinas com base nesses dados.

Outro problema que os simuladores analisados também possuem é a ausência de dados sobre disciplinas equivalentes, que permitem que mudanças no currículo do curso sejam adaptadas para corresponder com o currículo atual, por exemplo. Entretanto, diferente dos dados sobre pré-requisitos, os dados sobre disciplinas equivalentes não são sempre fornecidos pelos sistemas da universidade de maneira prática, como no caso da UFBA. Logo, embora seja de grande auxílio, não há como todas as ferramentas suportarem a exibição desses dados sem poder acessá-los. No entanto, mesmo em universidades que disponibilizam esses dados, como no caso da UFSC, as respectivas ferramentas que as atendem - como o **GRAMA**, o **MatrUFSC** e o **MatrUFSC2** - não possuem suporte a exibição desses dados, que poderiam auxiliar os estudantes no período da rematrícula.

Além disso, um outro aspecto que é um problema em boa parte dos simuladores analisados é a falta de um mecanismo para salvar o plano criado no servidor do simulador, ainda mais se for considerado o aspecto de segurança desse recurso. A falta dessa funcionalidade é um problema visto que o processo de criação de um plano de matrícula normalmente não envolve apenas uma única iteração de busca pelas disciplinas concluída rapidamente, mas vários acessos sendo realizados ao site em busca de atualizações tal como mudança de horário e professor das turmas em que o aluno tem interesse, além do recurso facilitar o compartilhamento do plano com outras pessoas. Uma observação importante é que entre as ferramentas que que permitem o salvamento dos planos no servidor (**GRAMA**, **MatrUFSC**, **MatrUFSC2**, **MatrUSP** e a nova versão do **MatrUSP**)

apenas o **MatrUFSC2** possui proteção contra sobrescrita, impedindo que o usuário salve o plano sem estar autenticado no site e impedindo que dois usuários, em duas contas diferentes, possam provocar conflito no identificador usado.

A falta desse recurso também impede a disponibilização de outro recurso que possui grande demanda entre os estudantes, que é a criação de planos considerando os planos salvos por diferentes amigos simultaneamente. Hoje, os alunos precisam se organizar para poder encontrar combinações que favoreçam horários que permitam aos mesmos cursarem as mesmas turmas, de forma que possam estudar juntos a mesma disciplina e assim terem mais facilidade nos estudos (incluindo o desenvolvimento de trabalhos e também a resolução de dúvidas do conteúdo da disciplina). Entretanto, esse é um trabalho extremamente manual, que envolve no melhor dos casos planilhas com as disciplinas que cada aluno pode realizar, porém ainda esbarrando em questões como validação de pré-requisitos e conflitos de horários.

Outro problema comum aos simuladores de matrícula analisados é o fato de que nenhum tem suporte genérico a mais de uma universidade. Ou seja, cada simulador tem como foco atender apenas uma única universidade (normalmente aquela em que o aluno que criou o simulador estuda) e o suporte a outras universidades só é possível no caso do sistema como um todo ser totalmente modificado (e não apenas uma pequena parte como opções em um banco de dados). Isso é bem ilustrado pelo **MatrUSP**, simulador criado para atender a USP e que é baseado no **MatrUFSC** (este criado para atender a UFSC), mas que possui modificações a nível de código para poder atender outra universidade, no caso um *web scraper* diferente para captura dos dados e uma tabela de horários adaptada para os horários da universidade desejada.

Existem problemas encontrados que atrapalham um pouco a usabilidade de alguns simuladores, como a ausência de busca de disciplinas, que afetam o **Grade na Hora!** e o **Meu Horário**. Isso faz com que o estudante precise verificar manualmente qual disciplina adicionar dentre uma lista que não é necessariamente pequena, mesmo quando considerando apenas as disciplinas de um curso, complicando portanto sua utilização em dispositivos móveis. Outro problema encontrado entre as ferramentas que fornecem busca - com exceção ao **MatrUFSC2** - é a ausência de suporte a correção de erros de digitação na busca, o que novamente dificulta a busca em dispositivos móveis, onde erros de digitação são mais frequentes e também em casos onde o aluno não sabe exatamente como digitar uma palavra.

Um problema encontrado no **Grade na Hora!** e que afeta bastante o uso de uma ferramenta desse tipo é a falta de um mecanismo de combinações, que faz com que o estudante precise verificar, manualmente, diferentes combinações entre as turmas das disciplinas escolhidas para encontrar um horário que lhe agrade. Outro problema, mais comum, é a falta de suporte a filtrar as combinações encontradas usando diferentes

Nome	Universidade	*Backend*	C1	C2	C3	C4	C5	C6	C7	C8	C9
GRAMA	UFSC	-	N	N	N	S	N	S	S	S	N
MatrUFSC	UFSC	Python	N	N	N	S	N	S	S	S	S
MatrUFSC2	UFSC	Python	N	N	N	S	N	S	S	S	S
MatrUSP	USP	PHP Python	N	N	N	S	N	S	S	S	S
MatrUSP 2	USP	PHP Python	N	N	N	S	N	S	S	S	S
GDE	UNICAMP	PHP	N	S	S	N	N	S	S	-	N[13]
Meu Horário	UFBA	PHP Ruby	N	N	N	N	N	S	S[14]	S	S
Meu Horário 2	UFBA	Ruby	N	S	N	N	N	S	N	N	N
Grade na Hora!	UTFPR	-	N	N	N	N	N	N	N	N	N

Tabela 1 – Comparação entre os simuladores de matrícula existentes.

critérios, como restrição de professores a serem ignorados no cálculo e também horários extra-curriculares, que não podem ser portanto ser ocupados por uma turma.

Dados todos esses problemas, podemos considerar que embora os simuladores sejam de grande auxílio para a maioria dos estudantes, é necessário recorrer a uma quantidade grande de recursos para poder realmente conseguir um horário adequado para o período letivo seguinte. Além disso, essas ferramentas são, em sua maioria, plenamente usáveis em um computador dado sua habilidade multi-tarefa já muito desenvolvida. Entretanto, em um dispositivo móvel o uso de ferramentas do tipo já começa a ser mais complexa, tendo em vista a sua limitada capacidade multi-tarefa como também os diversos problemas que esses dispositivos possuem, como problemas de conexão, tela pequena e afins. Com o fato dos dispositivos móveis serem hoje mais usados em relação ao acesso a internet quando comparado a um microcomputador no Brasil [IBGE 2015], esse tipo de problema são de fato empecilhos na busca por um plano ideal, e portanto não podem ser simplesmente ignorados.

A Tabela 1 apresenta uma análise comparativa entre os simuladores de matrícula apresentados neste capítulo. Os critérios comparativos considerados na tabela são descritos a seguir:

- **C1** - Suporte genérico a diferentes universidades;
- **C2** - Exibição da grade de pré-requisitos;
- **C3** - Sugestão de disciplinas baseado em pré-requisitos;
- **C4** - Salvamento de planos no servidor;

[13] Considera versão atualmente em produção
[14] É possível definir restrições de horários, mas sem definir qual a atividade a ser feita.

- **C5** - Suporte a navegação *offline*;
- **C6** - Atualização automática de disciplinas a partir do sistema da universidade;
- **C7** - Suporte a cadastro de atividades extra-curriculares;
- **C8** - Suporte a criação de combinações a partir das disciplinas escolhidas;
- **C9** - Sistema possui código aberto?

Observe que todos os sites mencionados usam apenas Javascript puro como tecnologia de *frontend*, sem envolver nenhuma linguagem que compilam para Javascript. Além disso, todos os sites mencionados também usam CSS puro, ou seja, não usam linguagens que compilam para CSS.

3 Estudo Preliminar e Levantamento de Requisitos

A proposta do presente trabalho é desenvolver um simulador de matrículas denominado **Guru da Matrícula** que seja capaz de realizar simulações de matrícula levando em consideração fatores tais como os pré-requisitos que o aluno já fez e também condições impostas pelo aluno tais como restrições de horários, de turmas e de professores, por exemplo. Em sua implementação final, o simulador fornece recursos para permitir a realização de combinações entre os planos de diferentes pessoas, enquanto possuindo capacidade para lidar com diferentes universidades e baixo consumo de recursos.

A plataforma deverá funcionar bem em dispositivos móveis, contando com suporte a navegação *offline* dos dados, além de uma interface adaptada e responsiva para uso tanto em dispositivos móveis quanto em microcomputadores, a partir de princípios do *Material Design* [Mew 2015].

3.1 Levantamento de Requisitos

Para o levantamento dos recursos mais importantes para o projeto, foi realizado uma pesquisa entre os alunos de diferentes universidades promovida através de redes sociais que possibilitou levantar quais os recursos mais desejados entre os estudantes - provido uma lista de recursos sugeridos e permitindo aos alunos o voto em vários recursos e também a entrada de outras sugestões - além de conhecer os problemas enfrentados pelos mesmos durante o período de rematrícula.

Essa pesquisa foi respondida por exatamente 100 pessoas, compostas por estudantes de diferentes universidades e também por ex-estudantes. Dentre essas pessoas, 44% responderam que não trabalhavam, 33% informaram que trabalham fora da universidade e apenas 23% indicaram trabalhar em algum laboratório ou estágio fornecido ou administrado pela própria universidade em que estudam.

3.1.1 Dificuldades para encontrar o horário adequado

Durante a pesquisa, foi questionado aos estudantes quais as **principais dificuldades** em encontrar um horário adequado para o período letivo seguinte. Como resposta, os estudantes podiam selecionar uma ou mais opções entre uma lista pré-determinada de respostas. A lista de respostas em questão está destacada abaixo:

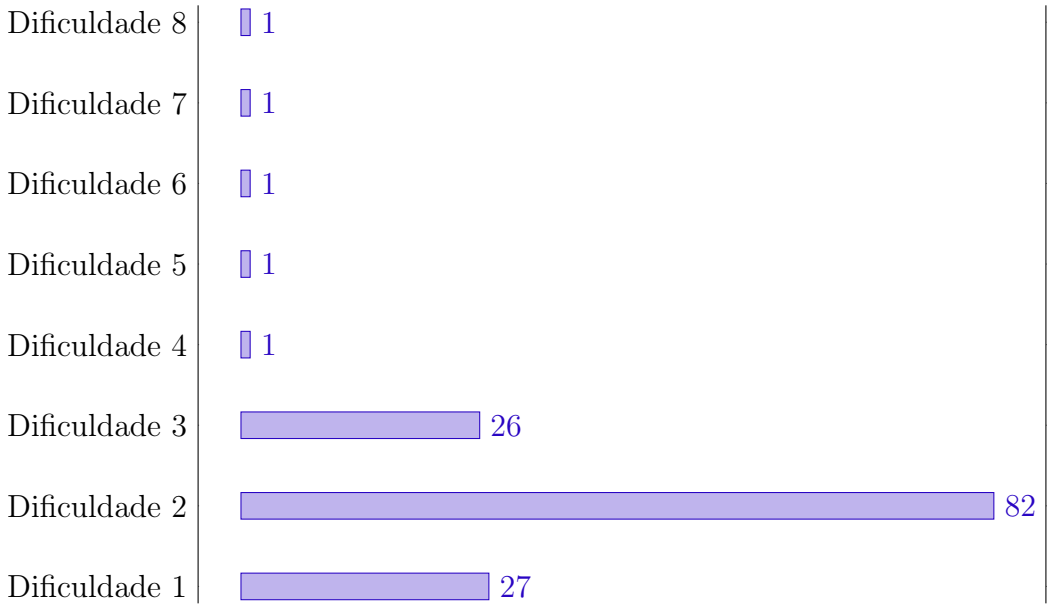

Figura 9 – Dificuldades para encontrar o horário adequado.

- **Dificuldade 1**: Encontrar disciplinas cujas dependências eu já cumpri;
- **Dificuldade 2**: Encontrar turmas no qual o horário é adequado às minhas necessidades;
- **Dificuldade 3**: Encontrar turmas em que meus amigos também estarão;
- **Dificuldade 4**: Horários vagos;
- **Dificuldade 5**: Não encontro dificuldade;
- **Dificuldade 6**: Acumular as aulas em blocos pra precisar ir menos vezes à Universidade;
- **Dificuldade 7**: Disciplinas ofertadas num campus de mais fácil acesso;
- **Dificuldade 8**: Informações sobre os professores.

A Figura 9 apresenta o resultado da enquete sobre as principais dificuldades encontradas ao encontrar um horário adequado. Como é possível observar, a maior parte dos votos estão concentrados quase que completamente nas três primeiras respostas. De acordo com as respostas, cerca de 82% dos alunos tem dificuldade em encontrar turmas cujo horário seja adequado as suas necessidades (como turmas em um só período ou turmas cujo horário não conflite com atividades extra-curriculares, por exemplo). Já cerca de 27% dos estudantes responderam que uma das principais dificuldades é encontrar disciplinas cujas dependências (ou pré-requisitos) já foram cumpridas. Enquanto que 26% responderam que uma das principais dificuldades é encontrar turmas em que seus amigos também estarão.

Houve também algumas respostas extras, como a de um estudante que respondeu não ter dificuldade e a de outro que respondeu que uma das principais dificuldades para encontrar o horário adequado é encontrar disciplinas cujas aulas sejam agrupadas em blocos tal que seja menor a frequência com que se vai à universidade. Este é um problema comum especialmente para quem mora longe da universidade em que estuda e enfrenta grandes congestionamentos no caminho até a universidade.

3.1.2 Recursos mais desejados pelos usuários

Durante a pesquisa, foi mostrado ao usuário uma lista dos recursos que ele mais gostaria de ver em um simulador de matrículas. Dessa lista, o usuário poderia selecionar múltiplos itens e adicionar itens conforme desejasse. A lista final de recursos e suas respectivas quantidades de votos estão referenciados na lista abaixo:

- **Recurso 1**: Aplicativo para dispositivo móvel;
- **Recurso 2**: Notificações em tempo real sobre mudanças em disciplinas;
- **Recurso 3**: Busca de disciplinas por professores;
- **Recurso 4**: Suporte *offline* (uso da plataforma como um aplicativo totalmente disponível sem conexão);
- **Recurso 5**: Suporte à localização de salas para auxílio durante a criação dos planos (por exemplo, para poder detectar quando horários muito próximos ficam em salas muito distantes);
- **Recurso 6**: Sugestão contextual de disciplinas por curso (com análise de dependências entre disciplinas e afins);
- **Recurso 7**: Busca por acrônimos dos nomes das disciplinas;
- **Recurso 8**: Exportação em formato *iCalendar*[1] para sincronização com aplicativos de calendários;
- **Recurso 9**: Área colaborativa de conteúdos sobre a universidade, processo de matrículas e afins;
- **Recurso 10**: Acompanhamento de disciplinas gerenciadas por professores;
- **Recurso 11**: Recursos sociais na plataforma, tais como notificações em tempo real sobre mudanças nos planos de amigos, criação de planos com maior número de disciplinas em comum com os amigos e suporte a *feedback* de determinadas combinações;

[1] O *iCalendar* é um formato que permite usuários enviar convites de reuniões ou tarefas para outros usuários através de compartilhamento de arquivos desse formato através de diversos métodos.

- **Recurso 12**: Suporte a outras universidades;
- **Recurso 13**: Espaço para colocar outras atividades com horário fixo no simulador, tais como curso de idiomas e atividades recreativas.

A Figura 10 apresenta o resultado da enquete sobre os recursos mais desejados pelos alunos. Nessa pesquisa, foi possível observar que a maior demanda em relação aos simuladores de matrícula é por um aplicativo móvel para *smartphones*, sendo a opção escolhida por 59% das pessoas. Entretanto, desenvolver um aplicativo que funciona bem em todas as plataformas é algo complexo e que exige bastante esforço, uma vez que é necessário desenvolver um código distinto para suportar cada plataforma, além de ter custos envolvidos na publicação nas diferentes lojas de aplicativo. Felizmente, é possível implementar sites que efetivamente são tão eficientes quanto aplicativos, inclusive contando com suporte a uso em tela cheia de forma nativamente, a navegação *offline* e até mesmo acesso rápido via ícone no menu de aplicativos. Além disso, esta abordagem conta com maior suporte, pois o mesmo código pode ser executado também em *desktops* e *notebooks* e não é necessário recorrer a lojas de aplicativos.

Em segundo lugar, com 49% dos votos, está o suporte a notificação em tempo real de alterações nas disciplinas, que permitiria enviar uma notificação ao usuário de forma próativa no momento em que uma modificação qualquer na disciplina fosse detectada (como por exemplo mudança de salas, de horários ou até mesmo de professores). Esse recurso permitiria que os estudantes soubessem, em tempo real, de alterações nas disciplinas que tem interesse, em vez de ter que acessar o sistema periodicamente para verificar se houve mudanças ou não.

Já o terceiro recurso mais desejado, com 46% dos votos, é a busca de disciplina por professores, que permite que com uma simples busca pelo nome do professor seja listado todas as disciplinas que este irá ministrar ou já ministrou no passado. Esse recurso muitas vezes não é fornecido ou não é facilmente acessível nos sistemas disponibilizados pelas universidades, como no caso da UFSC, que não disponibiliza tal opção de busca no sistema de rematrícula, mas permite a realização dessa busca na página de cadastro de turmas. Além disso, trata-se de um recurso muito útil para o aluno descobrir as disciplinas optativas disponíveis, visto que dessa forma o aluno consegue acompanhar facilmente as disciplinas que serão ministradas pelo professor da área que o aluno prefira.

O quarto recurso mais desejado pelos estudantes, com 45% dos votos, é o suporte *offline* do uso da plataforma. Ou seja, permitir que o aplicativo seja usável (com a parcela de dados mais usada pelo estudante já carregada no próprio dispositivo) sem estar conectado à Internet. Esse recurso não é encontrado em nenhum dos simuladores pesquisados e exige bom planejamento, visto que cada navegador define seu próprio limite de armazenamento e os dispositivos móveis usados atualmente ainda não são capazes de armazenar muitos dados

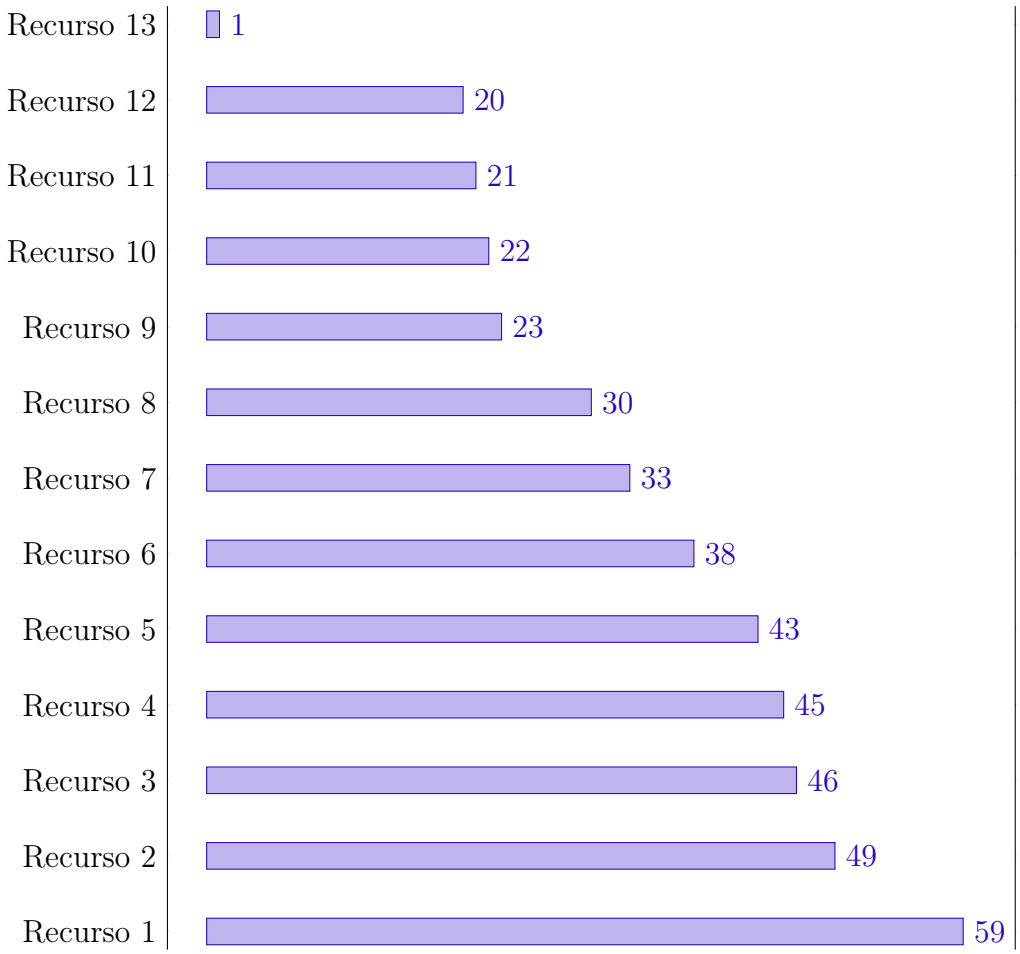

Figura 10 – Recursos mais desejados pelos alunos.

diretamente. Além disso, é necessário mecanismos para manter esses dados atualizados, o que pode aumentar o uso de recursos do servidor consideravelmente de acordo com o número de dispositivos onde os dados estão salvos para acesso *offline*.

O quinto recurso recebeu 43% dos votos e diz respeito a detectar a localização de cada sala visando evitar que o aluno precise caminhar muito entre uma sala e outra num curto período de tempo para ter aula, o que normalmente acontece quando as aulas são ministradas em salas localizadas em centros acadêmicos diferentes da universidade, que pode causar atrasos devido ao tempo para ir de um centro ao outro na universidade, por exemplo.

O sexto recurso foi votado por 38% dos alunos e diz respeito a sugestão de disciplinas baseada na análise de pré-requisitos. Esse recurso, como já explicado anteriormente, permite que o aluno já tenha, em primeira mão, uma lista das disciplinas que pode cursar visto que já cumpriu os pré-requisitos, o que facilita todo o processo de simular a matrícula. Além disso, dotado dos dados necessários para esse recurso (no caso, as disciplinas que o aluno já fez), é possível validar se uma disciplina tem ou não seus pré-requisitos respeitados,

podendo emitir tal aviso para o aluno imediatamente durante a simulação da matrícula.

O sétimo recurso recebeu 33% dos votos e se refere à busca de disciplinas por acrônimo, o que acontece em algumas disciplinas mas não em todas, como por exemplo "Sistemas Operacionais I" ser conhecida apenas por "SO I", por exemplo. Já o oitavo recurso, votado por 30% das pessoas, se refere ao suporte de aplicativos de calendários, como o *Google* Agenda, onde é possível cadastrar um *feed* em formato *iCalendar* e ter no calendário os dados de horários das aulas imediatamente sincronizado entre as plataformas.

O nono recurso mais desejado pelos alunos (com 22% dos votos) se refere a uma área colaborativa de conteúdos, onde seria possível que os alunos administrassem uma área com conteúdos, dentro da plataforma, relacionada à própria universidade onde estudam, com explicações a respeito do processo de matrícula, visando auxiliar calouros e alunos novos da universidade.

O décimo recurso mais desejado com 22% dos votos se refere a opção de "seguir" um determinado professor no sistema de forma a poder ver a acompanhar as disciplinas esse professor. Esse recurso, tal como a busca por professores, se mostra muito útil quando considerado a busca por disciplinas optativas, mas desta vez permitindo ao aluno saber em tempo real quais disciplinas optativas foram disponibilizadas para o próximo período letivo.

O décimo-primeiro recurso, com 21% dos votos, se refere à possibilidade de tornar a plataforma mais social, com recursos tais como detecção de planos com a maior quantidade de disciplinas e turmas em comum com os amigos e a notificação em tempo real sobre mudanças que estes venham a fazer em seus planos. Esse recurso tem como foco ajudar estudantes que criam seus planos buscando turmas em que seus amigos também estejam, visando facilitar o estudo para prova e também a realização de trabalhos em grupo que esta disciplina venha a possuir.

O décimo-segundo recurso, com 20% dos votos, se refere ao suporte do sistema por outras universidades. Esse recurso foi, entre os propostos oficialmente, o que recebeu menos votos dos usuários. Entretanto, esse resultado já era esperado visto que os estudantes realmente não precisam do suporte a outras universidades, mas sim do suporte à universidade em que estudam. Na prática, entretanto, projetar o sistema de tal forma que seja simples suportar outras universidades permite que o alcance da plataforma aumente, e que os outros recursos que a plataforma venha a suportar alcancem ainda mais usuários, mostrando a importância desse recurso.

Por último, um dos estudantes que respondeu a pesquisa sugeriu a colocação de um espaço para entrada de outras atividades (ou seja, atividades extra-curriculares) no simulador, tais como curso de idiomas, por exemplo, pois permitiria que a plataforma automaticamente ignorasse turmas que conflitam com as atividades extra-curriculares do

aluno. Observe que esse recurso foi o menos votado de todos por ter sido colocado por um usuário, ou seja, não aparecia para aos demais usuários da pesquisa. Entretanto, é um recurso muito útil pois permite ao aluno considerar todas as suas atividades enquanto faz o planejamento da matrícula, evitando assim eventuais conflitos durante o período letivo, que simplesmente não tem como ser detectados por um sistema acadêmico, por exemplo.

3.2 Definição dos requisitos do Projeto

Com base na análise dos resultados da pesquisa, na análise do contexto de duas universidades diferentes em relação ao sistema acadêmico e também nos simuladores de matrícula já definidos e apresentados no Capítulo 2, foi possível definir os requisitos funcionais do projeto de forma a melhor atender as necessidades dos alunos, que são os seguintes:

1. **Suporte a múltiplas universidades**: o sistema deve ser capaz de trabalhar com dados de mais de uma universidade, inicialmente UFSC e USP, com atualização automática dos dados;
2. **Salvamento de planos**: o aluno deve conseguir salvar planos no servidor, após efetuar *login*;
3. **Busca de disciplinas**: o aluno deverá conseguir buscar disciplinas em todas as universidades, campus e semestres cadastrados no sistema;
4. **Combinações em tempo real**: geração das combinações entre diferentes turmas em tempo real conforme o aluno adiciona e remove turmas e disciplinas do seu plano, mostrando visualmente os conflitos detectados quando nenhuma combinação puder ser gerada;
5. **Cadastro de disciplinas já realizadas**: usuários autenticados poderão cadastrar as disciplinas já feitas e receber sugestões de disciplinas que pode fazer baseado no cumprimento dos pré-requisitos;
6. **Planos compartilhados**: o usuário poderá criar planos compartilhados com os amigos, de forma a poder obter combinações onde turmas em comuns são priorizadas, de forma que o aluno possa estudar essas disciplinas com os amigos durante o período letivo;
7. **Acesso *offline***: o aluno deve conseguir baixar dados de cursos para acesso *offline* através do próprio sistema;
8. **Busca de professores**: O aluno deve conseguir ver todas as turmas que um professor ministra;

9. **Localização das salas**: o sistema deve ser capaz de mostrar os dados de localização da sala quando esses dados estiverem disponíveis.

Frente a esses requisitos, se tornou claro a existência de três atores que acabam por se tornar usuários do sistema:

- **Estudante Anônimo** - É o tipo mais comum de estudante, realiza acessos rápidos ao site para poder fazer checagens rápidas de disciplinas e possíveis combinações antes de se tornar um Estudante Autenticado;
- **Estudante Autenticado** - Procura por mais recursos, como salvamento de planos e sugestões baseadas em pré-requisitos, se tornando um usuário mais assíduo do site em relação ao Estudante Anônimo;
- **Universidade** - É a responsável primária pelo fornecimento dos dados do site. Se mostra como fator importante na relação entre quantidade de universidades suportadas e quantidade de estudantes usando a plataforma;

Dito isso, a Figura 11 mostra os requisitos da plataforma em relação aos usuários do projeto através de um Diagrama de Casos de Uso em *Unified Modeling Language* (UML) [Rumbaugh, Jacobson e Booch 2004].

Além dos requisitos funcionais, o projeto também possui alguns requisitos não funcionais que o permitirão:

- Consumir menos que 512MB de RAM por servidor (ou seja, no *backend*) durante seu funcionamento;
- Ser compatível com as duas versões mais recentes dos navegadores Google Chrome, Mozilla Firefox, Microsoft Edge e Apple Safari, incluindo também as versões móveis de cada navegador.

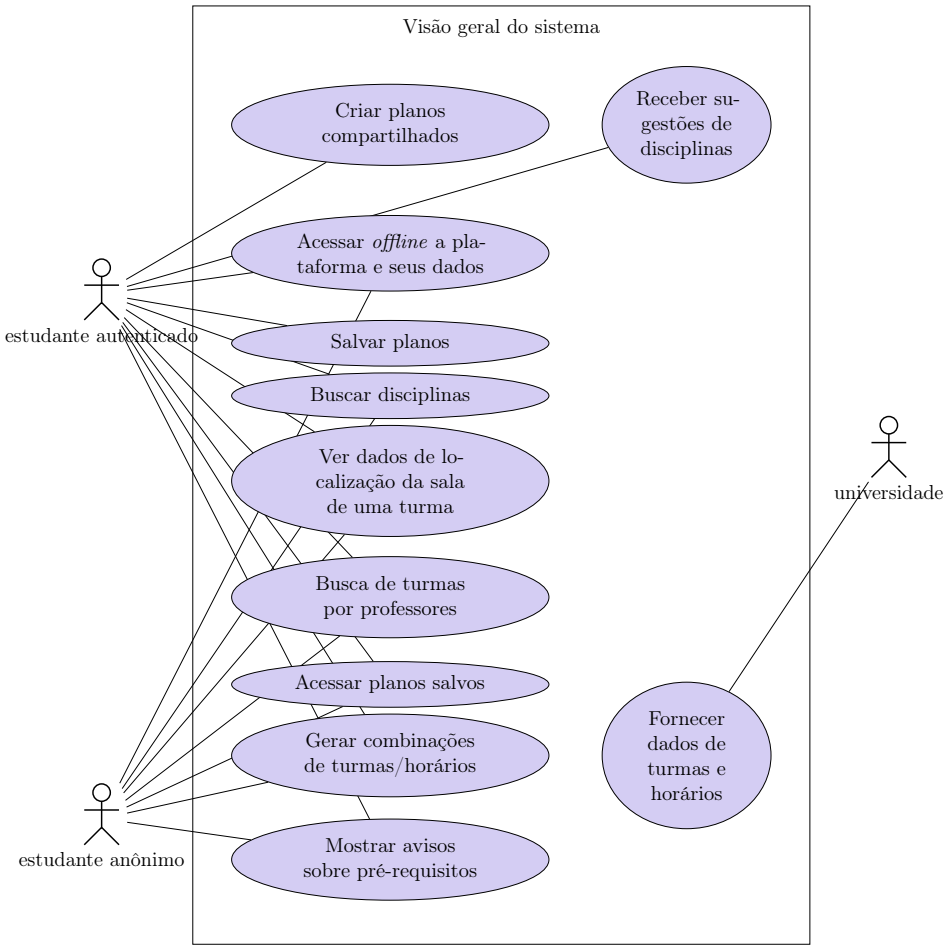

Figura 11 – Diagrama de casos de uso do projeto.

4 O protótipo: Guru da Matrícula

O projeto do **Guru da Matrícula** foi desenvolvido em duas partes separadas, mas integradas uma com a outra. De um lado, o *backend* será responsável por tratar os dados recebidos do usuário e das universidades, além de ser responsável pela busca e pelo fornecimento de dados coletados ao usuário. Do outro lado, o *frontend* será responsável por disponibilizar ao usuário uma interface para a visualização desses dados e também para o gerenciamento de planos em qualquer dispositivo (móvel ou não).

Cada uma dessas partes foi desenvolvida de uma maneira específica. O *backend* foi explicitamente desenvolvido focando em baixo consumo de memória e funcionamento distribuído, de maneira resiliente a falhas. Já o *frontend* foi projetado de forma a transferir a mínima quantidade de dados para o usuário, mas ainda fornecendo recursos como sincronização de dados para uso *offline* e interface adaptada para uso por dispositivos móveis.

O projeto pode ser acessado através do endereço: <https://gurudamatricula.fjorgemota.com/>.

4.1 Visão Geral

Por possuir uma série de requisitos complexos, o projeto foi dividido em várias partes, as quais foram integradas corretamente para que o sistema funcione. Numa visão global, é possível perceber uma divisão clara em duas partes: *backend* e *frontend*, desenvolvidas em linguagens diferentes e funcionando em conjunto para formar o resultado final. A Figura 12 mostra como é feita a integração entre o *frontend* e o *backend*.

Nesse modelo, o *frontend* é responsável por servir de interface para o usuário e fazer requisições ao *backend* em busca dos dados com o qual precisa trabalhar. Por exemplo, o *frontend* pode requisitar dados de uma determinada turma, de uma determinada disciplina, de um determinado professor, etc.

Por outro lado, o *backend* é responsável por organizar e otimizar os dados da aplicação, que são fornecidos tanto pelas universidades, na forma de turmas e habilitações para cadastrar, quanto pelos próprios usuários, na forma de planos salvos para um dado semestre. Além de tratar da organização e otimização do armazenamento, é o *backend* que atende as requisições de leitura para os dados armazenados, através do uso de índices de busca, *cache* e versionamento dos dados cadastrados.

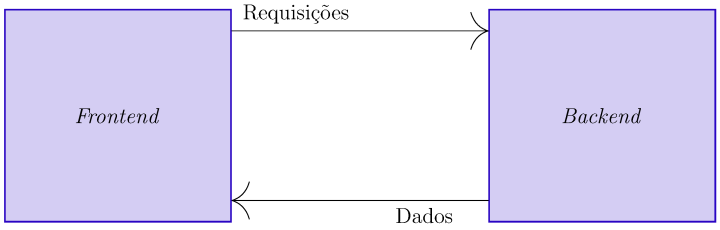

Figura 12 – Integração entre *backend* e *frontend*.

4.2 *Backend*

O *backend* foi dividido em duas partes principais: a aplicação, mostrada no bloco (2) da Figura 13, que contem toda a lógica de negócio do sistema, incluindo conhecimento sobre o formato das estruturas armazenadas no banco de dados e sobre o modelo de dados usado na comunicação com o *frontend* (que nada mais é do que um cliente do ponto de vista do *backend*, e portanto é representado no bloco (1)), e também os sistemas básicos, mostrado no bloco (3) que atuam como um *framework*, de forma genérica, fornecendo uma base simplificada e portável para a aplicação. Já os dados de turmas e cursos são fornecidos pelas universidades, representados no bloco (4), que transferem os dados para a aplicação a partir de arquivos JSON formatados de acordo com uma estrutura comum, de acordo com o tipo de dado a ser importado.

O *backend* foi projetado de forma a poder trabalhar com diferentes plataformas ao mesmo tempo que fornece um conjunto uniforme de recursos. De modo geral, o *backend* é responsável, entre outras coisas, por:

- Realizar o cadastro dos dados de turmas e horários a partir dos dados fornecidos

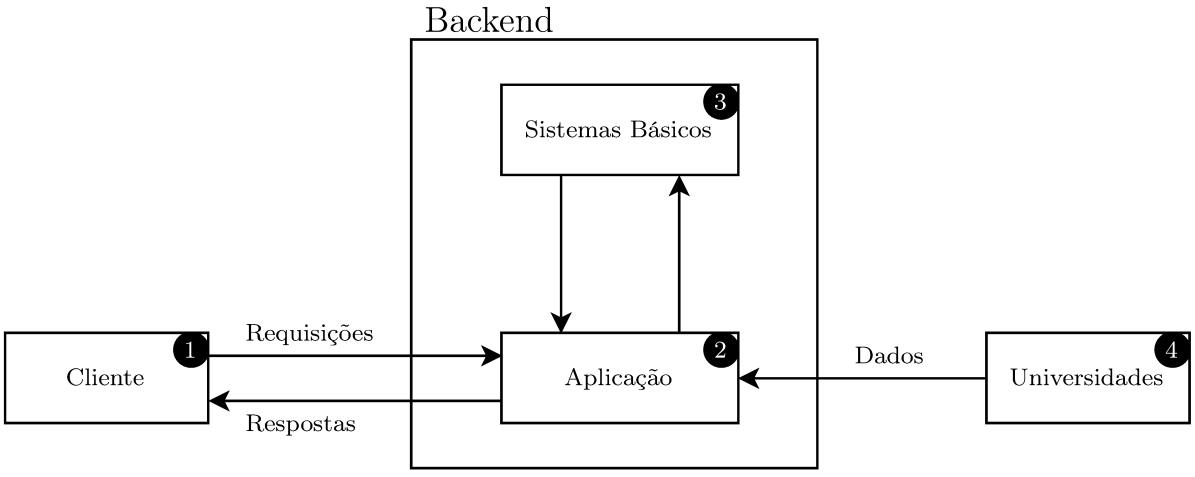

Figura 13 – Estrutura geral do *backend*.

pelas universidades (ou por *web scrapers*) periodicamente;

- Realizar autenticação (*login*/*logout*) do usuário;
- Armazenar os planos salvos pelo usuário associando com sua respectiva conta;
- Armazenar dados relacionados ao usuário tal como universidade em que estuda, curso que faz e disciplinas que já estudou;
- Realizar o versionamento dos dados disponibilizados e disponibilizar as mudanças em cada "versão" para cada curso, de cada universidade, de forma separada;
- Fornecer uma *Application Programming-Interface* (API) *GraphQL* para disponibilização dos dados da base de dados e também salvamento de dados de planos salvos por usuários;
- Realizar indexação e permitir busca dos dados armazenados através de diferentes filtros, tais como nome da disciplina ou professor que ministra as aulas.

Observe que, a princípio, o *backend* não é responsável por capturar e processar diretamente os dados das universidades (isto é, ele não busca interpretar o conteúdo das páginas de um sistema acadêmico). Em vez disso, o *backend* apenas executa requisições para programas separados, os quais podem ser desenvolvidos em linguagens diferentes, que então fornecem os dados em um formato adequado para o processamento do sistema, baseado no formato JSON mas adaptado para as estruturas de dados necessárias. Isso foi desenvolvido de forma que novas universidades possam ser adicionadas de forma simples, sem ser necessário conhecimento avançado do próprio funcionamento interno do *backend*, o que permite, inclusive, que a própria universidade forneça diretamente os dados, bastando apenas implementar o formato de dados JSON.

Nos casos das universidades que serão abordadas nesse trabalho (UFSC e USP), os dados serão capturados através de *web scrapers* desenvolvidos na mesma linguagem do *backend* que, a partir dos dados disponibilizados pelos sistemas acadêmicos, realizarão a conversão para o formato de dados JSON esperado pelo sistema. Tais mecanismos serão abordados na Capítulo 5.

4.2.1 Linguagens de Programação

O *backend* foi desenvolvido em **Go** [Pike 2009], uma linguagem de programação de código aberto desenvolvida pelo Google que tem como características o fato de ser concorrente, possuir compilação rápida e ter coleta de lixo (*garbage collection*). Esta linguagem possui grande semelhança com a linguagem **C** em termos de sintaxe, mas com a adição de comandos para programação concorrente e a remoção de parênteses em torno de estruturas como *for* e *if*.

Essa linguagem foi escolhida por possuir como características a capacidade de gerar binários estáticos como resultado da compilação. Além disso, possui baixo consumo de memória e é compatível com uma ampla variedade de arquiteturas, como ARM, x86, MIPS e afins, e também sistemas operacionais, como Linux, Mac OS X e Windows. A linguagem Go também possui uma grande biblioteca padrão e um grande ecossistema de ferramentas que fornece recursos como detecção de tratamento de erros, formatação do código (algo que a linguagem obriga nativamente ao compilar o código) e também detecção de possíveis *bugs*, facilitando assim o desenvolvimento de projetos de variados tamanhos.

Por fim, ela é uma linguagem capaz de aproveitar bem os processadores ou núcleos de processamento do servidor onde o programa é executado, pois ela permite dividir o trabalho entre inúmeras *goroutines* que são agendadas eficientemente pelo próprio ambiente de execução do Go onde o programa executa. Isso faz com que a linguagem seja perfeita para o desenvolvimento de programas que criam servidores *web* (como é o caso deste projeto), pois permite que cada requisição recebida seja tratada eficientemente e também que milhares de requisições possam ser processadas ao mesmo tempo. Por estas razões, a linguagem Go se torna mais eficiente do que outras linguagens interpretadas, tais como Python, PHP e Ruby.

4.2.2 Plataformas Suportadas

Devido à natureza multi-plataforma da linguagem de programação escolhida, é possível executar o projeto em todas as plataformas que o compilador suporta nativamente, bastando apenas fazer o *cross-compiling*[1] para essas outras plataformas.

Em relação às dependências da própria plataforma, como banco de dados e sistema de *cache*, o *backend* continua sendo compatível com as plataformas que o próprio Go suporta, mas também contando com suporte para outras plataformas, como é o caso do ecossistema do Google App Engine (GAE), o qual bloqueia escritas no próprio sistema de arquivos da máquina, incentivando assim o uso de serviços apropriados para as tarefas, como o *Google Cloud Storage*[2] e o *Google Cloud Datastore*[3].

Isso significa que o sistema será capaz tanto de executar em um microcomputador, quanto em um *Raspberry Pi* ou mesmo em um ambiente distribuído como o GAE, usando as APIs apropriadas através do uso de abstrações no *software*. Ou seja, através da implementação de um pouco de código, será possível fazer com que o sistema suporte outro Sistema Gerenciador de Banco de Dados (SGBD) facilmente.

Um aspecto importante a respeito dessas abstrações é que elas não tem um uso global sobre todo o sistema. Ou seja, elas são injetadas nos componentes que precisam delas

[1] Compilação para uma plataforma que não é a mesma onde o compilador está executando.
[2] *Google Cloud Storage* fornece armazenamento e leitura de arquivos com redundância e confiabilidade.
[3] *Google Cloud Datastore* é um banco de dados NoSQL altamente distribuído e escalável.

durante a inicialização desses componentes, o que permite maior flexibilidade e eficiência ao fazer que componentes diferentes possam usar dependências diferentes conforme suas necessidades.

4.2.3 Desenvolvimento de Testes

O *backend* possui grande parte do seu funcionamento verificado por testes unitários. Esses testes são executados a cada modificação, na própria máquina do desenvolvedor, e usam abstrações em memória dos sistemas básicos permitem testar a maior parte do código sem a necessidade de dependências externas. Dentro do sistema básico, cada abstração será testada de acordo com requerimentos a serem cumpridos, ou seja, o código relacionado ao GAE será testado caso os testes executem no ambiente de desenvolvimento fornecido pelo próprio GAE, por exemplo.

Tais testes foram desenvolvidos de forma a poder executar paralelamente em uma máquina, podendo assim aproveitar melhor a capacidade da máquina e melhorar o desempenho. Além disso, os testes estão disponíveis junto ao próprio código de cada pacote do sistema, por uma convenção da própria linguagem Go, o que ajuda a testar cada unidade do sistema individualmente e facilitando, portanto, o desenvolvimento. Por fim, será gerado um relatório de cobertura de código, que será usado para demarcar quais trechos do código não foram testadas por nenhuma função e também permitir acompanhar a evolução dos testes a cada *commit*.

4.2.4 Sistemas básicos

A aplicação depende de alguns sistemas básicos, cuja estrutura está destacada na Figura 14, que fornecem a base para o funcionamento do sistema, como uma espécie de *framework*. Entre os serviços fornecidos, estão acesso a banco de dados (bloco (7)), *cache* (bloco (10)), gerenciamento de arquivos (bloco (3)), indexação de dados e busca (bloco (4)), fila de tarefas (bloco (9)), *pipeline* de tarefas (bloco (6)), entre outros, menos importantes mas ainda úteis, como o relógio (bloco (11)), um reciclador de memória (bloco (5)) e alguns utilitários para testes (bloco (8)), que atuam de forma genérica, sem conhecimento interno das características do resto da aplicação (representado no bloco (2)), e que podem ser usados de forma também genérica, com o uso de adaptadores permitindo a substituição dos componentes sem a necessidade de refatorar todo o código desde que implementado as interfaces desses sistemas básicos.

Graças a esses sistemas básicos é que é possível obter, além de portabilidade, maior performance, ao facilitar a implementação de algoritmos mais eficientes. Um exemplo é a importação de dados das universidades (representadas no bloco (12)), que é uma tarefa realmente intensiva em termos de quantidade de dados, e que é facilmente paralelizada

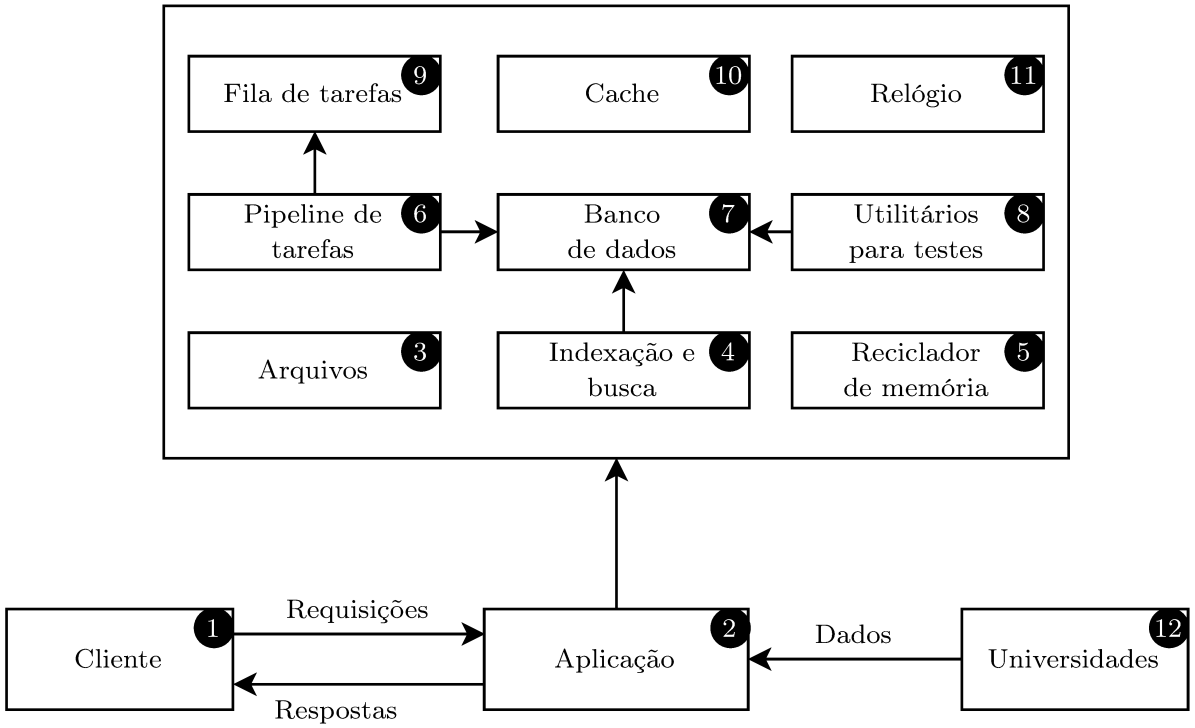

Figura 14 – Estrutura geral dos sistemas básicos.

com o uso da *pipeline* de tarefas (representada no bloco (6)), que é utilizado pela aplicação (bloco (2)) para diminuir o tempo necessário para importar os dados. Além disso, outro exemplo é o atendimento à requisições do cliente (bloco (1)), que se aproveitam bastante tanto da implementação de *cache* (bloco (10)), quanto do sistema de indexação e busca (bloco (4)).

Essas características permitem que esses sistemas básicos possam ser usados em outros projetos e obter essas mesmas características de portabilidade e foco em performance que este sistema possui.

4.2.4.1 Banco de Dados

O banco de dados é um dos sistemas básicos mais importantes fornecidos. Dentre as suas características, está o fato de implementar uma interface *tabela-chave-valor*, onde, dado um conjunto de uma tabela e uma chave, é possível tanto salvar quanto obter o valor salvo no banco de dados, além de permitir a iteração por tabelas inteiras, ter suporte rudimentar a filtragem (sem uso de indíces) e transações.

Isso é fornecido através de uma interface de alto nível implementada por variados adaptadores, que permitem que diferentes bancos de dados sejam usados pelo sistema de forma invísivel a este. Dentre as interfaces implementadas, temos:

- ***AppEngine*** - Adaptador para a *Google AppEngine Datastore*;
- ***Badger*** - Adaptador para o Badger, que implementa um SGBD *embedded* no processo do sistema, otimizado para SSDs e muito otimizado para escritas;
- ***Bolt*** - Adaptador para o Bolt, que implementa um SGBD *embedded* no processo do sistema, com o uso de arquivo mapeado em memória. É otimizado para leituras;
- ***File*** - Adaptador que usa o sistema básico de arquivos para salvar os dados em arquivos. Útil para testes;
- ***Datastore*** - Adaptador para o *Google Cloud Datastore* e *Google Cloud Firestore*;
- ***Memory*** - Adaptador que salva os dados na memória RAM, e que não é persistente (útil para testes);
- ***Cached*** - Adaptador que, dado outro adaptador e uma instância de algum *Cache*, armazena as leituras e escritas no cache, de forma a acelerar as operações

4.2.4.2 Cache

O *cache* é um sistema básico desenvolvido para fornecer *performance* extra nas operações de leitura ao realizar o armazenamento, de maneira temporária, de dados. Assim como o sistema básico de banco de dados, o *cache* opera com uma interface única acompanhado por vários adaptadores, que implementam essa interface e permitem o uso de diferentes técnicas e sistemas de *cache*.

Uma característica importante que diferencia o sistema básico de *cache* do sistema básico de banco de dados é o fato de que, além dos dados do *cache* serem totalmente temporários, o sistema de *cache* é modelado em uma interface simples de *chave-valor*, o que simplifica a implementação. Além disso, o sistema de *cache* não possui suporte à iteração sob as chaves armazenadas. Dentre as implementações de *cache* disponíveis, temos:

- ***AppEngine*** - Permite o uso da implementação *Memcached* do AppEngine como *cache*;
- ***LRU*** - Implementa o adaptador de *cache* armazenando os dados em uma estrutura com LRU, onde os dados mais antigos não-usados vão sendo apagados em favor dos mais recentemente usados;
- ***Multi*** - Adaptador que permite o uso de múltiplos adaptadores de *cache*;
- ***Nil*** - Adaptador que não salva dado algum, apenas descartando todos os dados salvos e sempre retornando erro ao obter registro.

4.2.4.3 Arquivos

O sistema básico de arquivos é desenvolvido para fornecer uma interface para um sistema de arquivos, independente de ser um sistema de arquivos local ou remoto. Diferentemente de um sistema de arquivos completo, entretanto, esse sistema básico não administra pastas nem permissões (no sentido de bloquear um arquivo para escrita, por exemplo), ou seja, a interface só fornece acesso a arquivos, que são retornados como *streams* que podem ser lidos ou escritos (dependendo do método chamado), e também podem ser fechados.

Assim como os outros sistemas básicos, uma série de adaptadores que implementam a interface são implementados, como mostra a lista a seguir:

- ***Local*** - Adaptador que implementa a interface através do acesso ao sistema de arquivos local, através da API do sistema operacional no qual o sistema está sendo executado;
- ***GCS*** - Adaptador que implementa a interface do sistema básico de arquivos acessando a API do Google Cloud Storage, para uso no AppEngine;
- ***Memory*** - Adaptador que simula o armazenamento dos arquivos em memória, de maneira não persistente. Útil para testes;
- ***Multi*** - Adaptador que implementa a interface através do uso de múltiplos outros adaptadores, sendo apenas um usado para escrita e um ou mais sendo usado para leitura. Útil para testes.

4.2.4.4 Fila e pipeline de tarefas

Esses dois sistemas básicos cumprem uma tarifa fundamental para a aplicação: são elas que permitem que o sistema trabalhe de maneira distribuída, gerenciando as tarefas e todo o grafo de dependências entre tarefas de maneira automática, o que permite que um conjunto de tarefas seja disparado e, assim que todas as tarefas terminem, o sistema dispare uma outra tarefa para tratar os resultados, por exemplo, entre outras possibilidades bastante interessantes que são exploradas na aplicação.

No caso desses dois sistemas, o mais básico é o de fila de tarefas, que possui a função de armazenar a fila de tarefas e dispará-las, seguindo restrições como numero de tarefas concorrentes sendo executadas e intervalo entre tarefas. Além disso, esse sistema possui uma restrição simples: as tarefas disparadas devem ter sempre como alvo um *endpoint* HTTP, de forma que uma tarefa chamada possa estar em outro servidor, por exemplo.

Esse sistema possui duas interfaces básicas com apenas quatro adaptadores, no total. A primeira interface define os disparadores de tarefas, e os adaptadores que a implementam

são três: local, que dispara as tarefas usando *goroutines*, *AppEngine*, que usa a API de *taskqueue* do AppEngine para disparar tarefas, e *Cloud Tasks*, que usa a API do *Google Cloud Tasks* para fazer a mesma tarefa. Já a segunda interface é o expedidor de tarefas, que tem como objetivo armazenar, codificar e decodificar chamadas com seu respectivo conjunto de argumentos e é implementado por um único adaptador, que compartilha código com o expedidor de tarefas do *pipeline* de tarefas.

Já o *pipeline* de tarefas, que é um sistema mais avançado comparado ao de fila de tarefas, conta com acesso também ao banco de dados além da fila de tarefas, e tem como objetivo realizar o controle e administração de todo o grafo de tarefas. Dessa forma, cada nova tarefa vira um registro no banco de dados e são registrados também todos os filhos da tarefa, as tarefas dependentes e as dependências. A ideia desse sistema é permitir realizar todo o controle das tarefas agendadas e executadas de forma que a aplicação possa lidar apenas com uma API de alto nível, sem precisar administrar questões como ordem e dependência das tarefas, por exemplo.

Uma vantagem proveniente do uso desse sistema é a paralelização automática de tarefas de acordo com as dependências da tarefa. Isso significa que, se uma tarefa dispara outras X tarefas que não possuem dependências, todas essas X tarefas serão agendadas para execução paralela e executadas respeitando o limite de tarefas concorrentes. Isso também significa que, se uma tarefa Y depende do resultado dessas X tarefas, somente após o final da execução de todas as X tarefas a tarefa Y será executada.

4.2.4.5 Indexação e busca

O objetivo do sistema básico de indexação e busca é permitir a criação de índices inversos para os itens do banco de dados, de forma que seja possível buscar pelos itens de forma exata, em que apenas os resultados que coincidem completamente são retornados, por diferentes chaves, ou ainda de maneira *full-text*, ou seja, com suporte à busca por prefixos, *auto-complete* e correção de erros de digitação.

O sistema, por si só, possui algumas interfaces principais, e dois adaptadores que implementam, cada uma, um tipo de indice: Índice Exato, e Indice *full-text*, conforme descrição acima. O sistema possui suporte aos seguintes recursos e características:

- **Atualizações incrementais** - Permitem inserir, atualizar ou remover itens do indice sem reconstruir o índice completamente;
- **Suporte a reconstrução do índice** - Permitem criar uma nova versão, totalmente do zero;
- **Versionamento** - Permitem acessar diferentes versões, que são imutáveis, de um mesmo indíce de modo a ter resultados consistentes entre diferentes requisições;

- **Suporte a otimização** - Gera uma nova versão do índice composta por todas as atualizações incrementais feitas até então, de forma a gerar uma única versão otimizada para acesso;
- **Coleta de lixo** - Detecta e apaga dados de versões antigas que não são mais referenciadas no sistema, de forma a liberar espaço em disco;
- **Suporte a consultas complexas** - Permitem a realização de buscas complexas, com suporte a operadores lógicos tais como *AND*, *OR* e *NOT*.
- ***Sharding*** - Possibilita o agrupamento de vários registros em um só, de forma a permitir o carregamento de vários registros em uma só operação, além de permitir melhor compactação dos dados;
- **Compactação de índice inverso** - Permite economizar espaço e obter boa performance mesmo em consultas complexas através da compactação do índice inverso com usando *Roaring Bitmaps* [Lemire et al. 2018].
- **Compactação de árvore de prefixos e correções** - Permite que o índice inverso *full-text* possa atender consultas por prefixo ou consultas com erros de digitação usando o mínimo espaço, memória e processamento possível através do uso de uma *Deterministic Acyclic Finite State Automaton* (DAFSA) [Daciuk et al. 2000] para armazenar as palavras que pertencem ao índice;

Como limitações, o mecanismo não suporta atualizações concorrentes para um mesmo índice, e não realiza *parsing* das consultas a partir de entradas do usuário, o que significa que a estrutura lógica da consulta precisa ser repassada ao mecanismo de índice já na forma de estruturas de dados representando a consulta desejada.

4.2.4.6 Outros sistemas básicos

Além dos sistemas básicos tratados acima, a aplicação faz uso de alguns outros sistemas básicos importantes, mas menores. Alguns desses sistemas básicos são, inclusive, usados apenas para situações específicas, como testes, por exemplo. Dentre os sistemas básicos citados abaixo, os mais usados são o Relógio, que permite abstrair o acesso à hora atual ao mesmo tempo que facilita a implementação de testes, e o Reciclador de memória, que ajuda a otimizar o funcionamento do sistema ao fornecer ferramentas para diminuir a carga sob o *Garbage Collector* do Go.

- **Relógio** - Permite acesso ao horário atual e à funções como *Sleep*, para fazer o programa aguardar determinado numero de segundos. Também fornece abstração para uso em testes, permitindo testar sistemas que são muito dependentes do horário;

- **Utilitários para testes** - São utilitários variados úteis apenas para testes, que permitem, entre outras coisas, salvar o conteúdo das requisições feitas em um banco de dados e usar esses dados para emular o acesso à internet, de forma muito rápida e eficiente, além de permitir a administração de arquivos temporários para testes, incluindo, também, descompactação automática dos dados;

- **Reciclador de memória** - Permite reaproveitar certas estruturas afim de diminuir a carga sobre o *Garbage Collector* do Go, mas sem garantir sua persistência ao longo da execução do programa. Fornece utilitários como para reciclagem de *buffers*, estruturas de *cache* para operações comuns como conversão de inteiros para *string* e afins.

4.2.5 Aplicação

A aplicação é dividida em 3 partes principais: uma para realizar o controle de operações relacionadas a usuários (bloco (4) da Figura 15) - como cadastro e autenticação - outro para realizar a importação dos dados (bloco (5)) das universidades (bloco (6)), e, por fim, outro para permitir o acesso a esses dados pelo cliente, através de uma API pública *GraphQL* (bloco (3)).

Além disso, a aplicação também possui uma parte auxiliar, que é responsável por atender as requisições do cliente (bloco (1) da Figura 15) e direcioná-las para o sub-sistema correto, e é denominada roteador (bloco (2)),

Este sistemas são dependentes de abstrações extras que usam os sistemas básicos (bloco (7) da Figura 15) apresentados na Seção 4.2.4 para, entre outras coisas, realizar o processamento e a persistência dos dados. São essas abstrações que garantem a consistência e a padronização dos dados salvos, permitindo que o dado escrito possa ser lido em um

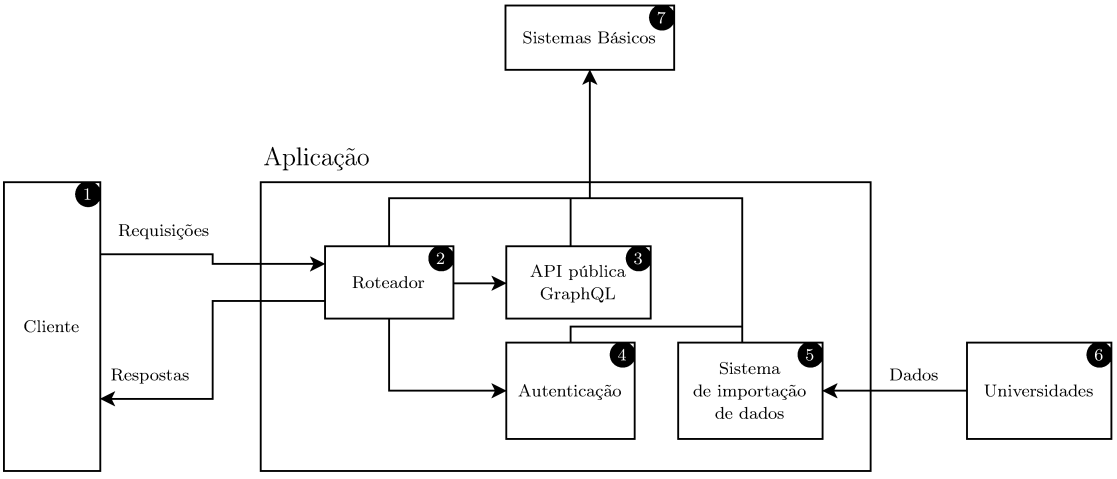

Figura 15 – Estrutura geral da Aplicação.

mesmo formato otimizado para armazenamento em banco de dados, por exemplo.

4.2.5.1 Sistema de autenticação

O sistema de autenticação é, basicamente, uma série de estruturas integrando a biblioteca *Authboss*[4] (bloco (2) da Figura 16) com o resto da aplicação (bloco (3)) e fornecendo persistência a partir do sistema básico de banco de dados (bloco (4)), de forma a fornecer, entre outros recursos, cadastro, *login* e *logout* para o cliente (bloco (1)).

Através desta biblioteca, o sistema de autenticação provê suporte a cadastro, autenticação e *logout*, tanto a partir do tradicional conjunto de *e-mail* e senha quanto a partir da integração, via OAuth2, com sites como Google e Facebook. Dados importantes, como senhas, são codificados usando **BCrypt**, o que garante a segurança desses dados em caso de invasão ou vazamento dos dados do banco de dados, por exemplo.

A interface externa a esse sistema é uma API *Representational State Transfer* (REST) básica, disponibilizada de forma a permitir, entre outras coisas, a integração com os serviços OAuth2 e a possibilidade de confirmação de e-mail. Essa API é exposta diretamente pelo *Authboss*, e usa o formato JSON para comunicação básica com o cliente, além do uso de redirecionamentos *Hypertext Transfer Protocol* (HTTP) sempre que necessário.

Dentre os recursos disponiblizados por este sistema, estão:

- **Cadastro e autenticação por OAuth2** - Fornece integração com Google e Facebook;
- **Cadastro e autenticação por *e-mail*/senha** - Permite o cadastro de pessoas que não tem interesse em usar Google ou Facebook para autenticação;
- **Recuperação de senhas** - Opção disponibilizada para quem preferir autenticação por usuário/senha;
- **Confirmação de conta** - Exigido para quem se cadastra sem usar a integração com Google ou Facebook. Garante que o usuário que está se cadastrando realmente tem acesso ao e-mail informado;

[4] <https://github.com/volatiletech/authboss>

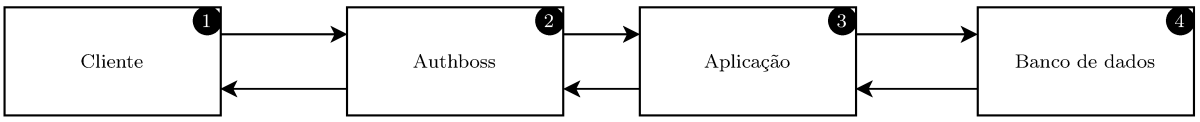

Figura 16 – Estrutura do sistema de autenticação

Além de prover os recursos acima, esse sistema também é usado para prover autorização a determinados recursos dos outros sub-sistemas da aplicação, como o cadastro de planos e de universidades (expostos a partir da API pública *GraphQL*). É a partir dessas checagens de autorização, por exemplo, que o sistema garante que um usuário não pode editar dados do plano ou universidade cadastrados por outro usuário, o que garante a segurança do sistema.

4.2.5.2 Sistema de importação de dados

O sistema de importação de dados é um mecanismo distribuído, composto por componentes conectados entre si que realizam, de maneira otimizada, a inserção, atualização e remoção de dados do banco de dados (representado pelo bloco (8) da Figura 17) a partir dos arquivos JSON criados seguindo um padrão comum (que varia de acordo com o tipo de dado a ser importado, que pode ser tanto dados de turma, quanto dados de habilitações) a partir dos dados disponibilizados pelas universidades (bloco (1)). Ou seja, esse sistema, por si só, **não lida** com a extração dos dados das universidades. Em vez disso, é esperado um arquivo JSON padronizado, cujo *download* é realizado na etapa (3) do processo de importação.

Embora seja distribuído, o sistema não suporta atualizações concorrentes para um mesmo tipo de dado, graças à necessidade de lidar com arquivos da última importação realizada com sucesso (demonstrado no bloco (5) da Figura 17) para poder detectar, no bloco (4), as diferenças entre cada arquivo, ou seja, o que foi de fato inserido, atualizado ou removido entre os arquivos, afim de poder fazer a atualização dos dados no banco de dados e também no índice de busca, nas etapas (6) e (7).

Devido a essa limitação, há uma etapa, representada no bloco (2) da Figura 17,

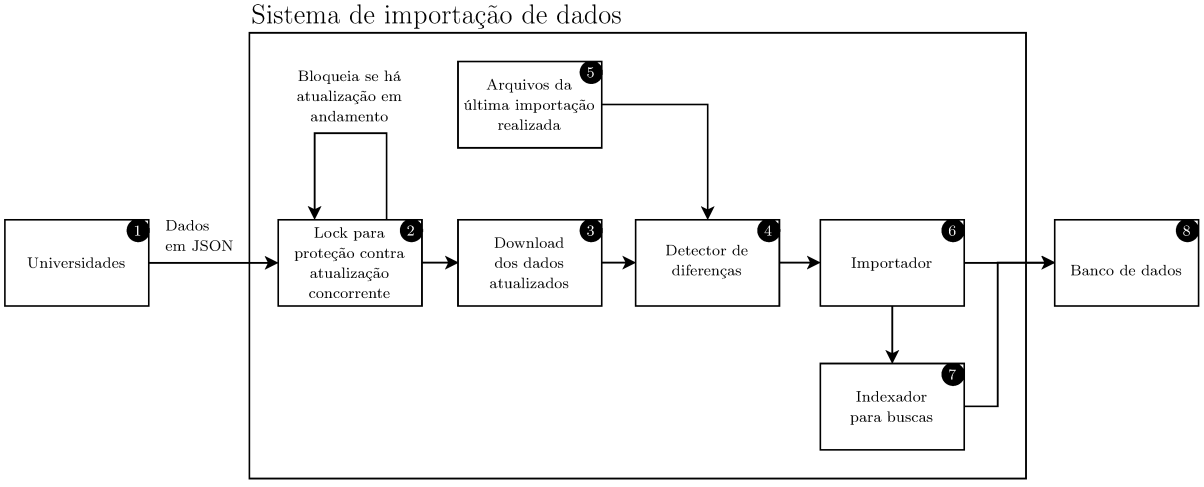

Figura 17 – Estrutura geral do sistema de importação de dados.

onde é realizada uma checagem para garantir que não há nenhuma tarefa de importação para uma dada universidade e tipo de dado (no caso, se é oferta de turmas ou se são dados de cursos) executando no momento, através de um *Lock* distribuído usando os aspectos transacionais do sistema básico de banco de dados. Só quando a tarefa até então em execução terminar é que é disparado um novo processo de atualização, o que garante tanto a consistência do processo de importação quanto alivia a carga sob o sistema.

As etapas (6) e (7), mostradas na Figura 17, são realizadas de maneira paralelas de acordo com a própria natureza das atualizações. Tabelas são atualizadas de maneira totalmente paralela, por exemplo, dado que cada registro emitido pelo detector de diferenças se refere a exatamente um registro da tabela, que pode ser inserido, atualizado ou removido. Já os índices são, cada um, atualizados de maneira sequencial, com as remoções acontecendo primeiro, registros novos sendo inseridos em seguida e os registros modificados sendo atualizados por último, mas índices diferentes são processados de maneira paralela conforme possível. Todos esse processamento, no final, acaba resultando em operações para o sistema básico de banco de dados.

Ao final do processo, o registro da própria universidade processada é atualizado com dados como última atualização e lista de arquivos da importação recém-atualizada. Além disso, o "dono" da proteção contra atualização concorrente é apagado, o que permite que a próxima tarefa de importação seja executada logo na sequência.

4.2.5.3 API pública *GraphQL*

A API principal do sistema, que permite o acesso a todos os dados coletados pelo sistema de importação, é exposta sob a interface da linguagem de consulta *GraphQL*. A única parte não totalmente exposta sob *GraphQL* é a API de autenticação, que depende de *endpoints* próprios para permitir autenticação *OAuth2* (ou seja, com Facebook e Google), entre outras ações.

A API *GraphQL* é exposta a partir de um modelo pré-definido, que define todos os tipos, campos, métodos e seus argumentos que podem ser usados nas consultas em *GraphQL*. Para criar a API, foi utilizado o projeto **gqlgen**, representado no bloco (2) da Figura 18, que gera código Go baseado no modelo *GraphQL*, e que auxilia na "tradução" das requisições GraphQL em chamadas à aplicação, que então consulta o banco de dados (bloco (5)) e retorna os dados, que são codificados de acordo com o padrão *GraphQL*, para o cliente (bloco (1)).

Quando a requisição *GraphQL* é somente-leitura, a aplicação, em vez de consultar diretamente o banco de dados, faz as requisições usando um *dataloader* (bloco (4) da Figura 18), que, entre outras propriedades, provê um *cache* temporário (apenas durante a duração da requisição) e permite agrupar diferentes consultas paralelas ao banco de dados (na mesma requisição) em uma só consulta, o que permite reduzir a latência de

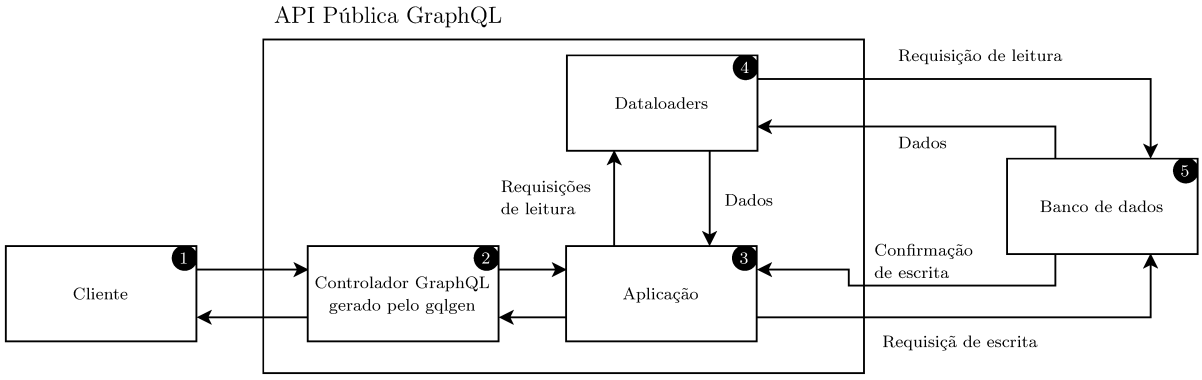

Figura 18 – Fluxo de atendimento de consultas *GraphQL*.

comunicação com o banco de dados em SGBD que suportem a leitura de múltiplos registros em uma só consulta.

Quando a requisição *GraphQL* é uma mutação, ou seja, altera algum registro do banco de dados, tal otimização não é possível, e então a aplicação faz a requisição de atualização diretamente ao banco de dados, como mostra a Figura 18, com a conexão direta entre os blocos (3) e (5).

Em ambos os casos, fica sob responsabilidade do controlador gerado pelo *gqlgen* a validação dos tipos da requisição - de acordo com o definido no modelo *GraphQL* - e fica sob a responsabilidade da aplicação a validação final da requisição, através da verificação da autorização do usuário para fazer a requisição, por exemplo, de forma a impedir que usuários possam modificar ou mesmo ler registros sob os quais não tem as respectivas permissões.

Em termos de performance, o fato do *gqlgen* ser um gerador de código permite que o compilador da linguagem otimize o resultado final, o que compensa parcialmente o custo do *parsing* das requisições *GraphQL*. Além disso, se um usuário adiciona mais de uma consulta à uma única requisição *GraphQL*, tais consultas são executadas paralelamente, o que ajuda a diminuir o tempo de resposta consideravelmente.

4.3 *Frontend*

O *frontend* do **Guru da Matrícula** foi desenvolvido de forma a ser leve (menor do que 100KB no carregamento inicial) e responsivo entre vários dispositivos. Essa parte do sistema é a parte visível ao usuário e funciona apenas em navegadores recentes, usando recursos baseados em HTML5 [Hickson et al. 2014]. Dentre as características do *frontend*, estão:

- Visual baseado em *Material Design* [Mew 2015];
- Suporte a *download* de código sob demanda;
- Salvamento de dados para acesso *offline* usando *Service Workers* [Russell et al. 2017] e *IndexedDB* [Alabbas e Bell 2017], com suporte a sincronização de mudanças incrementais;
- Geração de combinações de turmas sob demanda, com suporte à filtragem das combinações geradas;
- Emissão de avisos tais como falta no cumprimento de pré-requisitos e conflitos de horários;
- Interface para administração de contas do usuário, incluindo opções para configurar o curso que o usuário faz, o campus e universidade em que estuda, pré-requisitos já cumpridos, e configurações relacionadas a salvamento de dados *offline*.

O *frontend* foi desenvolvido e testado primariamente em celulares e então adaptado para funcionamento com dispositivos que possuem telas maiores, tendo sido testado exaustivamente para permitir sua correta execução a 60 Quadros por Segundo (QPS), mesmo em celulares, mantendo a fluidez e responsividade no uso. Por fim, o *frontend* é um *Progressive Web App* (PWA) [Ater 2017], ou seja, uma aplicação capaz de se adaptar progressivamente aos recursos disponíveis no navegador do usuário. Na prática, isso quer dizer que a aplicação será:

- **Confiável**: carregamento instantâneo, independente das conexões de rede, após o primeiro acesso;
- **Rápida**: responde rapidamente as interações de usuário e sem travamento mesmo em ações simples como *scroll* de página;
- **Atraente**: similar a um aplicativo no dispositivo com experiência de usuário imersiva.

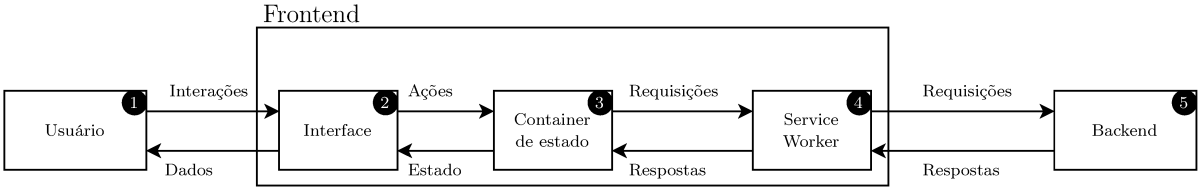

Figura 19 – Estrutura geral do *frontend*.

Para que tudo isso fosse possível, o *frontend* foi dividido em três partes principais, conforme mostrado na Figura 19: a **interface**, mostrada no bloco (2), com a qual o usuário (bloco (1)) interage e visualiza diretamente, o ***container* de estado**, mostrado no bloco (3), que contem a lógica das ações associada com a interface, e o ***Service Worker***, mostrado no bloco (4), que intercepta as interações entre a página (composta por interface e *container* de estado) e o *backend* (mostrado no bloco (5)) e permite que toda a aplicação funcione mesmo sem conexão com a internet, através do armazenamento de dados em um banco de dados local no navegador do usuário.

4.3.1 Linguagens de Programação

O *frontend* foi desenvolvido em **Typescript** [Jansen 2015], uma linguagem desenvolvida pela Microsoft que é um superconjunto com suporte a tipos do Javascript, linguagem muito usada para adicionar interatividade em páginas *web*. Como o Typescript compila para Javascript, o resultado final para o usuário será o mesmo e qualquer navegador com os recursos necessários e Javascript habilitado será capaz de executar o código.

A escolha pelo Typescript surge do fato de que o Javascript por si só é uma linguagem dinâmica e sem tipos. O Typescript adiciona suporte avançado a tipagem na linguagem, permitindo que muitas inconsistências sejam detectadas imediatamente durante a compilação do código e não apenas em tempo de execução (*runtime*). Além disso, como o resultado efetivo gerado é apenas o código Javascript sem os tipos anexados, o resultado final é o mesmo, não havendo custo adicional de processamento para o usuário devido ao uso da linguagem.

Em termos de estilo, o **Guru da Matrícula** usa *Sassy CSS* (SCSS), que é uma linguagem que compila para *Cascading Style Sheets* (CSS) [Jr., Rivoal e Etemad 2017], esta, sim, suportada por todos os navegadores modernos e que permite definir características básicas de estilo de texto, tais como posicionamento, cor de fundo e cor do texto. A vantagem do uso do SCSS para definição dos estilos é a integração com as bibliotecas de estilo e a maior facilidade de customização através do uso de variáveis e funções auxiliares (que são corretamente transformadas para CSS puro durante a compilação). Em relação às bibliotecas de estilo, o *frontend* usa uma biblioteca chamada *RMWC* em conjunto com a *material-components-web*, que implementa a linguagem de *design* chamada *Material Design*, o que permite ao sistema ter um visual respeitando essa linguagem de *design*.

4.3.2 Plataformas Suportadas

Em relação as plataformas suportadas, o *frontend* possui um conjunto limitado de navegadores suportados. Não serão suportados navegadores antigos e/ou abandonados, tais como Internet Explorer. Em vez disso, serão suportados apenas as duas últimas

versões estáveis de cada navegador, mas sem garantir acesso a recursos avançados como acesso *offline*. Felizmente, como os navegadores hoje contam com suporte a atualização automática, o número de pessoas usando navegadores atualizadas é bem grande. Segundo estatísticas globais publicadas pelo site *StatsCounter*[5], esse limite compreende mais de 60% do *marketshare* global de navegadores.

Além disso, graças ao fato de que esses navegadores suportam os recursos mais recentes em termos de Javascript e CSS, é possível diminuir muito o tamanho do código final entregue ao usuário ao evitar o uso de *polyfills* e *shims* criados para suportar navegadores antigos.

4.3.3 Desenvolvimento de Testes

O *frontend* foi desenvolvido de forma a, eventualmente, suportar testes unitários. Isso é feito através de uma implementação que não usa variáveis globais para armazenamento de estado e é muito idempodente, sendo totalmente dependente dos parâmetros passados. Graças a isso, e ao uso do *Typescript*, tem-se uma verificação inicial básica da corretude do código, que poderá ser melhorada no futuro com a implementação de testes verificando o comportamento do código escrito em outras plataformas e, também, situações extremas.

Dentre os testes feitos, estão o uso de uma ferramenta chamada *Lighthouse*[6], que ajuda a checar aspectos de performance da experiência do usuário na página, como velocidade de carregamento, de exibição da página e simulação do desempenho em dispositivos móveis.

Já a comunicação com o servidor é testada através do compartilhamento do modelo *GraphQL*, que permite que seja analisado, em tempo de compilação, a corretude do código comparado ao que o servidor entrega. Isso facilita bastante a implementação do sistema, uma vez que, ao conhecer a estrutura fornecida pelo *backend*, é possível trabalhar com dados estruturados de forma mais fácil e ainda contar com as checagens feitas pelo *Typescript*.

4.3.4 Interface

A interface foi desenvolvida usando *React*[7], que é uma biblioteca Javascript extremamente otimizada para desempenho baseada na ideia de *virtual-DOM*, onde o sistema retorna a estrutura desejada para o componente de acordo com um estado específico, e a própria biblioteca se encarrega de detectar as diferenças em relação ao que já está sendo exibido e, então, realizar apenas o conjunto mínimo de operações necessárias para que o componente exibida corresponda exatamente ao que a aplicação retornou.

[5] <http://gs.statcounter.com/browser-version-market-share>
[6] <https://developers.google.com/web/tools/lighthouse/>
[7] <https://reactjs.org/>

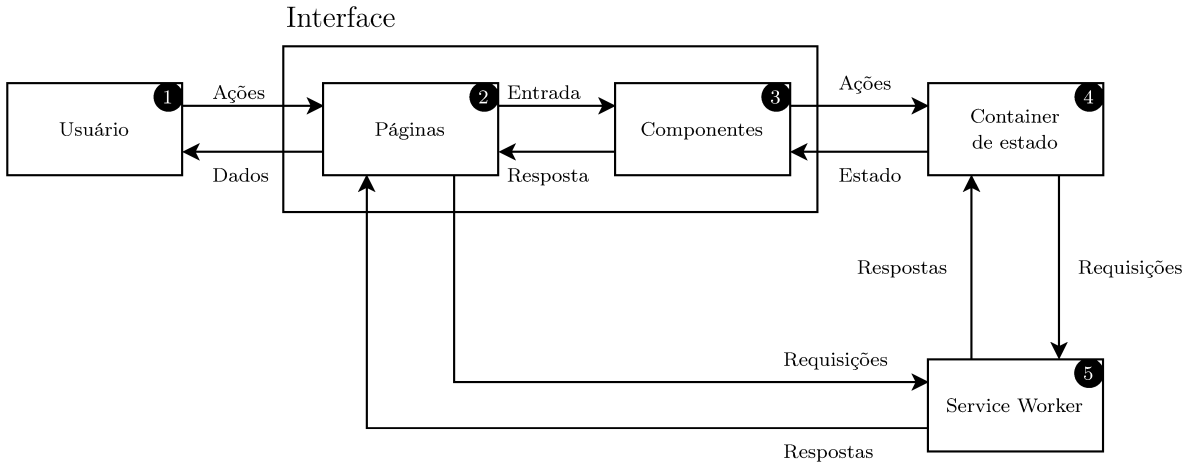

Figura 20 – Estrutura geral da *interface*.

Essa abordagem proporciona benefícios tanto em desempenho quanto em confiabilidade ao evitar que a aplicação precise manipular, diretamente, os elementos visuais correspondentes ao que está sendo atualizado, o que acaba reduzindo a complexidade da aplicação e portanto seu tamanho. Desta forma, é projetar componentes que são independentes do resto do aplicativo, e também mais fáceis de testar, uma vez que a estrutura retornada pelo componente depende apenas da entrada de dados e do estado interno (que tende a ser pequeno, por se tratar de apenas uma pequena parte da página), do próprio componente, que pode ser facilmente reproduzido em qualquer ambiente *Javascript*.

Em relação à própria estrutura interna da interface, os componentes são implementados de maneira a serem reusáveis sempre que possível, possibilitando reaproveitar, por exemplo, componentes que aparecem em mais de uma página do sistema, como calendários e menus laterais. Isso ajuda a diminuir a quantidade de código total necessária para renderizar a interface, além de simplificar o código e implementação.

Tais componentes são carregados de forma dinâmica, sob-demanda, dependendo da página acessada pelo usuário, o que ajuda a diminuir ainda mais o tamanho do código enviado para execução pelo navegador do usuário. Essas páginas, representadas no bloco (2) da Figura 20, são estruturadas como componentes especiais que, diferente dos demais componentes reusáveis (bloco (3)), são projetados para lidar com aspectos próprios do conteúdo a ser exibido, como consultas ao servidor (ou *service worker*, como mostrado no bloco (5)) e também leitura de parâmetros da URL, o que faz com que tais páginas operem de maneira mais específica e menos dependente do componente. Para ajudar na manutenção do estado entre os componentes, é utilizado um *container* de estado (bloco (4)) que permite sincronizar o estado compartilhado entre diferentes componentes, que também é capaz de realizar requisições diretamente para o *backend/service worker* (bloco (5)).

Por fim, todo o visual da interface foi baseado em *Material Design* [Mew 2015], implementado no código a partir de uma biblioteca chamada *RMWC*[8], que implementa componentes reusáveis respeitando as regras visuais do *Material Design* para *React*, através do uso da biblioteca oficial para *web*, chamada *material-components-web*[9].

Alguns detalhes adicionais sobre as páginas pertencentes à interface do sistema estão descritos nas seções abaixo:

4.3.4.1 Autenticação

As páginas de autenticação são, em sua maioria, páginas públicas, que fornecem ao usuário a opção de fazer cadastro, *login* e recuperação de senha. Por sua funcionalidade, essas páginas tem como característica estar centradas, principalmente, em formulários, que requisitam do usuário as informações que são necessárias para que a funcionalidade seja implementada. Junto a esses formulários, são mostrados também textos auxiliares escritos para guiar o usuário durante o processo, com informações que possam ajudá-lo a entender o foco do formulário, além de disponibilizar links para outras páginas úteis (por exemplo, ao permitir que o usuário acessando a página de *login* possa acessar a página de cadastro facilmente, e vice-versa).

A página de cadastro, mostrada na Figura 21, é uma página com um formulário contendo quatro campos: nome, *e-mail*, senha e confirmação de senha. A partir dessa página, é permitido ao usuário se cadastrar o conjunto de *e-mail* e senha, evitando portanto que o usuário precise ter contas em outros serviços e ainda fornecendo a possibilidade, ao usuário, de usar uma senha diferente para uso do serviço. Nessa página não é fornecido opções para cadastro usando serviços de terceiros pois, ao fazer o *login* pela primeira vez, caso ainda não cadastrado, o usuário usa a API do serviço a partir do qual o usuário se conectou para obter as informações necessárias para completar o cadastro, como nome e *e-mail*, por exemplo. Entretanto, são fornecidos links de acesso rápido às páginas de *login* e de recuperação de senha, para caso o usuário observe que não precisa, de fato, criar uma nova conta para acessar o sistema.

Já a página de *login*, mostrada na Figura 22, é uma das páginas mais complexas entre as páginas de autenticação. Nela, é possível fazer *login* a partir de qualquer uma das variadas opções de autenticação suportadas pelo sistema, que incluí o tradicional conjunto de *e-mail*/senha, assim como *login* através de serviços de terceiros, como Google e Facebook.

No caso desses serviços de terceiros, como é necessário abrir páginas externos ao site, tais páginas são abertas usando *pop-ups* que, então, permitem que o usuário entre com os dados para o site externo e, ao final do processo, retorne automaticamente ao

[8] <https://rmwc.io>
[9] <https://github.com/material-components/material-components-web>

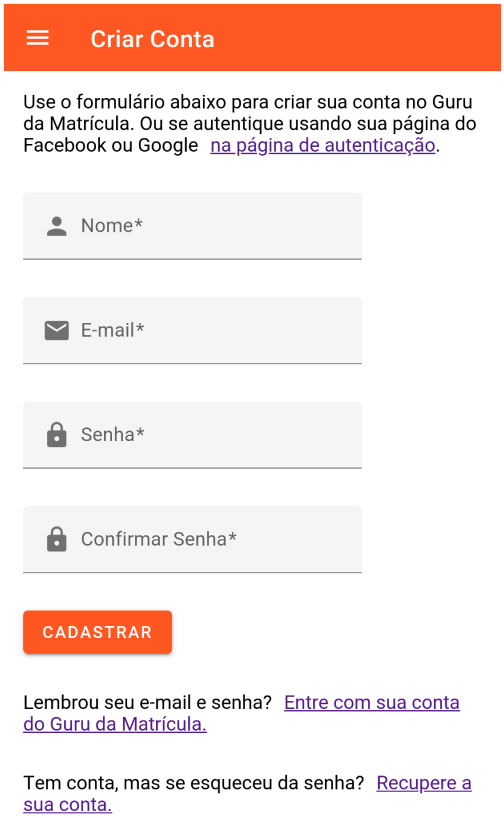

Figura 21 – Página de cadastro.

site, que detecta automaticamente problemas no processo. Por meio dessa opção de *login* senhas não são trocadas, o que garante a segurança do processo. Já no caso de *login* usando usuário e senha, ao enviar o formulário o sistema faz uma requisição AJAX para o servidor HTTP (especificamente para o endereço no qual o *Authboss* atende) com as credenciais e, em caso de erro, avisa o usuário. Em caso de *login* bem sucedido, o usuário é redirecionado para a página definida como parâmetro do endereço da página ou, caso nenhum parâmetro tenha sido especificado, para a página inicial do site. Em caso de falha uma mensagem é mostrada para permitir que o usuário tome providências e eventualmente repita o processo.

Por fim, a página de recuperação de senha, mostrada na Figura 23, é a página mais simples entre as páginas de autenticação, possuindo apenas um formulário com apenas um campo, para que o usuário entre com o *e-mail* usado no cadastro da conta, de forma que o sistema possa enviar um *e-mail* para esse usuário fornecendo um *link* especial que, quando clicado, permite que o usuário mude a senha usada na conta. Além desse formulário, também são fornecidos *links* de acesso rápido para as páginas de *login* e de *cadastro*.

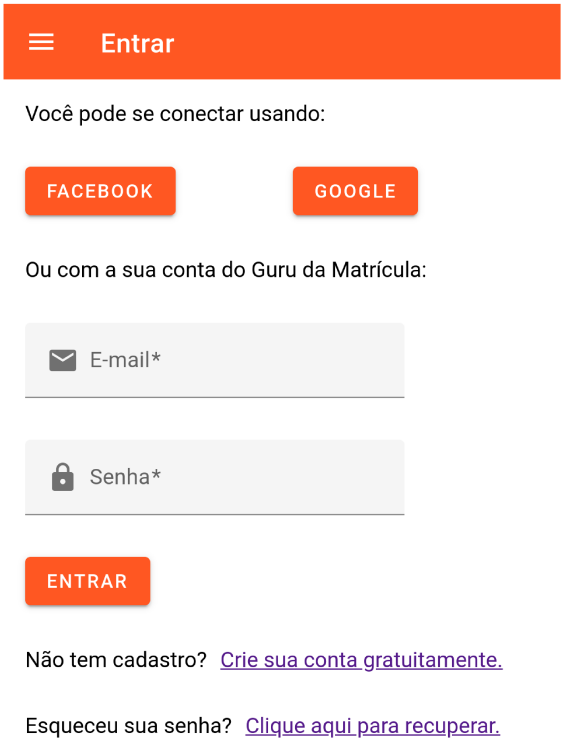

Figura 22 – Página de *login*.

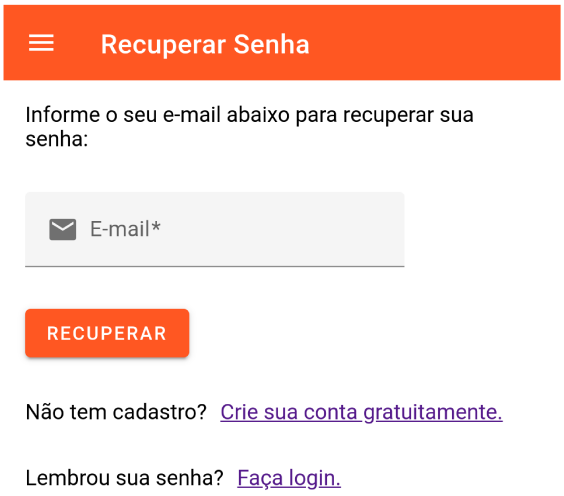

Figura 23 – Página de recuperação de senha.

4.3.4.2 Planos

As páginas de planos são as mais acessadas pelos usuários, pois tem como função permitir a administração de planos contendo todo o planejamento para um dado período letivo e universidade.

As duas páginas mais proeminentes dentre as páginas relacionadas a planos são: a página de calendário, mostrada na Figura 24, onde é possível visualizar todo o conjunto

Figura 24 – Página de calendário mostrando combinação encontrada de plano.

de horários e também as turmas escolhidas para uma determinada combinação, e a página de busca, mostrada na Figura 25, que permite adicionar novos registros (sejam disciplinas ou ofertas de disciplinas) ao plano. Ambas as páginas tem diferentes variações, de forma a permitir que o usuário possa explorar tanto diferentes dados (ao buscar por turmas ou buscar diretamente por disciplinas, por exemplo), como também visualizar tais dados de diferentes modos (ao visualizar apenas 1 ou 3 dias da semana, em vez de 7, no calendário, por exemplo).

Também há páginas para visualizar e gerenciar as disciplinas e ofertas de disciplinas adicionadas ao plano, junto a outras páginas com funcionalidades variadas, como permitir gerenciar configurações, como nome do plano, *status* de publicação e outras opções de configuração, ou ainda visualizar versões antigas do plano.

4.3.4.3 Universidades

As páginas de universidades são umas das menos acessadas pelos usuários, pois tem como utilidade permitir o gerenciamento das universidades cadastradas no sistema, algo que não deverá ser de grande interesse por parte dos usuários visto que o foco, e também a maior parte dos usuários, do sistema são estudantes. Esse procedimento inclui, entre outras coisas, ter conhecimento de duas tarefas básicas: como extrair os dados de turmas e cursos da universidade, e também como formatar todos esses dados no formato JSON, na

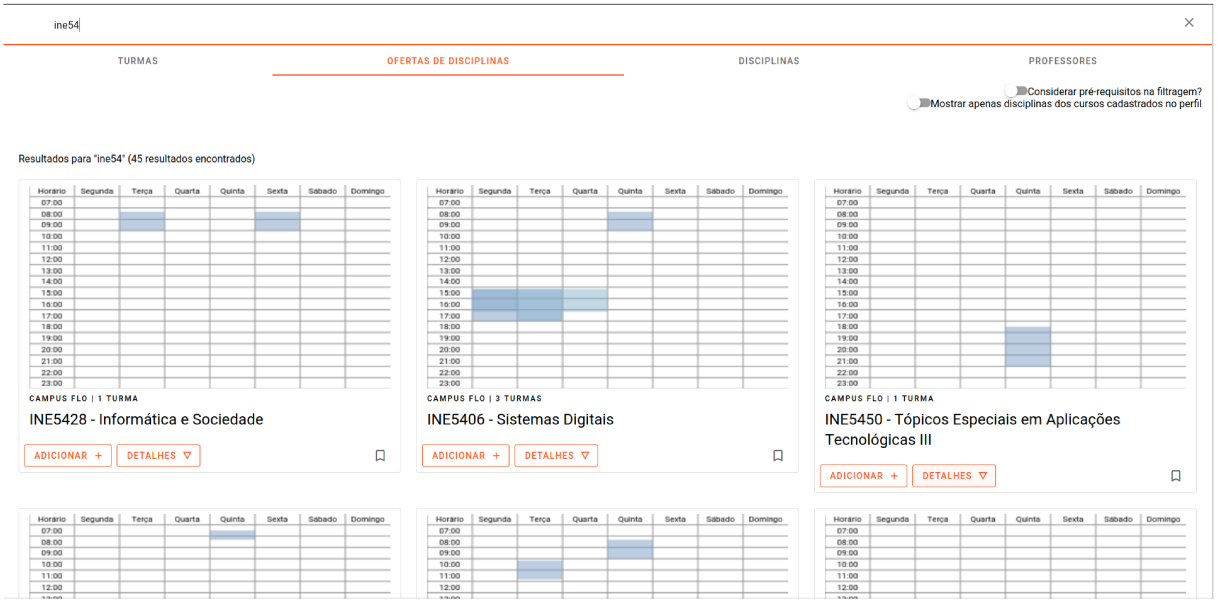

Figura 25 – Página de busca de ofertas de disciplinas.

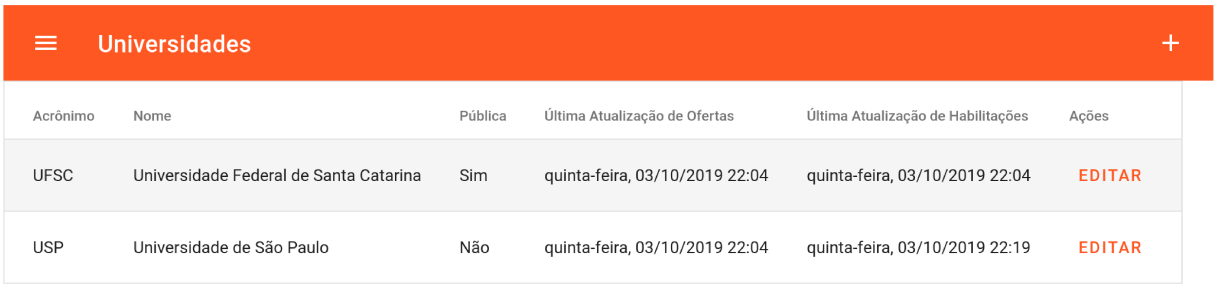

Figura 26 – Lista de universidades.

estrutura exigida pelo sistema de importação de dados, explicado na Seção 4.2.5.2.

A primeira página desse conjunto é a lista de universidades, mostrada na Figura 26, que tem como função listar todas as universidades que o usuário possuí, além de fornecer acesso rápido para as páginas de edição de universidade e também de cadastro de universidades. Nessa página também são mostrados dados associados a cada universidade, como data e hora da última atualização correspondente a cada universidade. Note que, após cadastrada, uma universidade não pode ser apagada, mas pode ser ocultada para os demais usuários.

Outra página importante é a página de cadastro de universidade, mostrada na Figura 27, que permite o cadastro de uma universidade no sistema. Para isso, são solicitados informações básicas, como nome e acrônimo, além de informações mais detalhadas tanto para dados de turmas quanto para dados de cursos, como endereço de download do arquivo

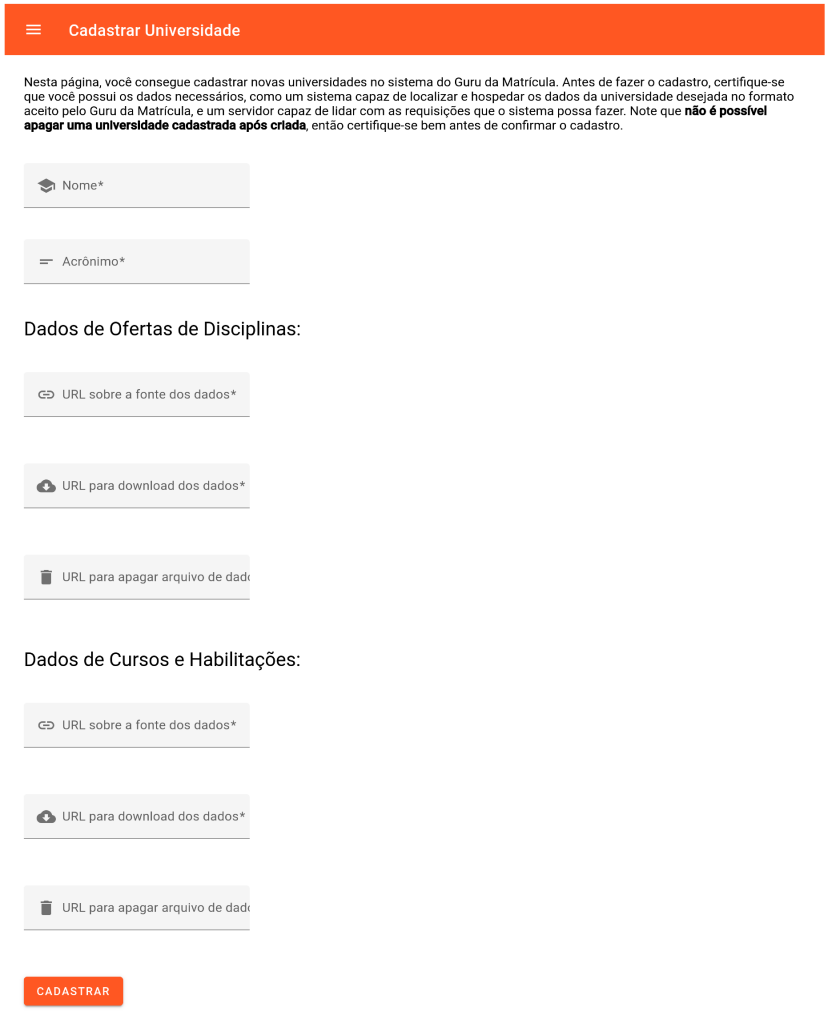

Figura 27 – Página de cadastro de universidades.

em formato JSON (formatados conforme o formato esperado), endereço para saber mais sobre os dados baixados e, também, endereço de notificação para apagar o arquivo dos dados originalmente baixados. Ao enviar o formulário, o sistema cadastra a universidade e retorna dois códigos secretos: um para dados de turmas, e outro para dados de curso. Esses códigos secretos devem ser usados para poder comunicar o sistema de importação de dados sobre a disponibilidade dos arquivos de dados para download, e não devem ser compartilhados, pois poderiam permitir a entrada de dados errados no sistema. A universidade cadastrada é salva com visibilidade oculta (não-pública), de forma que é possível entrar com os dados no sistema antes de finalmente disponibilizar a universidade para uso pelos demais usuários.

Por fim, a página de edição de universidade, é muito similar à página de cadastro de universidade, mas com apenas a adição de três campos: um para permitir que a universidade seja liberada para acesso pelas outras universidades, e outros dois para permitir que novos códigos secretos sejam gerados tanto para o envio de dados de turmas quanto para o envio

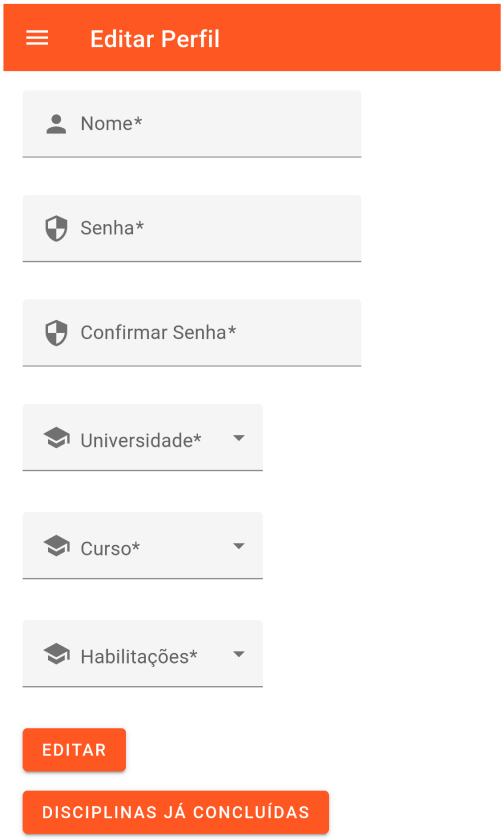

Figura 28 – Página de atualização do perfil do usuário.

de dados de cursos. Após o envio do formulário, são mostrados os respectivos códigos secretos solicitados, caso o campo em questão tenha sido marcado, o que garante que o usuário tenha a flexibilidade de mudar a chave de segurança sempre que desejar.

4.3.4.4 Perfil

A página de perfil, mostrada na Figura 28, é uma das páginas mais complexas e importantes do sistema. Sua função é permitir que o usuário possa definir dados relacionados ao seu próprio cadastro, como nome, *e-mail*, cursos, habilitações e também as disciplinas que já fez. Além disso, há a opção de configurar a senha, para usuários cuja conta foi criada usando o conjunto de *e-mail* e senha, uma vez que contas criadas a partir de serviços externos não podem ter senha devido a sua própria natureza de depender da autenticação pelo serviço externo em questão.

A página mais complexa relacionada às páginas de perfil é o de seleção de disciplinas já feitas, mostrada na Figura 29 que precisa permitir ao usuário lidar com uma grande quantidade de dados, e por isso faz consultas ao servidor visando economizar na quantidade de dados transferidos ao mesmo tempo que facilita a administração das opções por parte do usuário.

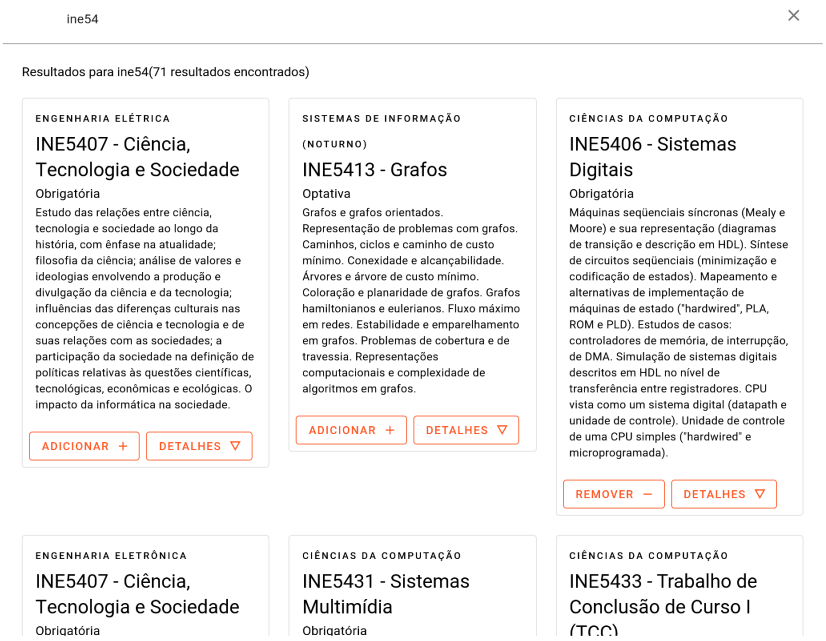

Figura 29 – Página de seleção de disciplinas já feitas.

4.3.5 Container de estado

O *container* de estado é constituído por um conjunto de partes que são carregados sob demanda de acordo com a página em que o usuário está e que, utilizando a biblioteca *Redux*[10], implementam um repositório responsável por armazenar o estado do *frontend*. Por usar *Redux*, tem-se que o estado é imutável, com novos estados sendo gerados de acordo com ações normalmente disparados por ação do usuário, computados por funções denominadas *reducers* (bloco (2) da Figura 30) que, dado o estado anterior e uma ação (que pode ser proveniente do usuário ou de qualquer outro sub-sistema), computam e retornam um novo estado.

Devido a isso, a lógica da aplicação fica protegida contra atualizações concorrentes e é, desta forma, consistente, facilitando tanto a implementação de testes quanto a eventual reprodução de *bugs* e outras falhas na lógica. Além disso, ganha-se acesso a ferramentas que possibilitam, entre outras coisas, a análise das ações registradas até então e também do estado do repositório após efetuar cada ação, o que facilita muito o desenvolvimento. Por fim, ao estruturar-se dessa forma, a lógica do *frontend* acaba por ficar isolada da camada de interface (representada no bloco (1) da Figura 30), o que significa que é possível mudar toda a interface - incluindo bibliotecas e estruturas - sem a princípio ser necessário alterar a lógica de estado do *container* de estado.

Junto ao *container* de estado são executadas funções *Sagas*, representados no bloco (3) da Figura 30, que são efeitos colaterais baseadas nas ações disparadas pelo usuário

[10] <https://redux.js.org/>

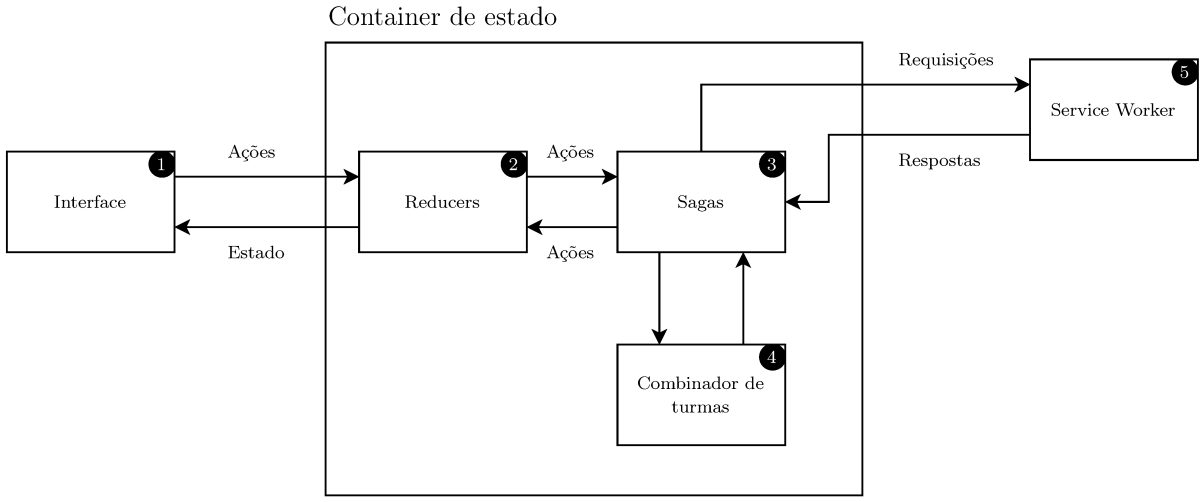

Figura 30 – Estrutura geral do Container de estado.

gerenciadas pela biblioteca Redux Saga[11]. Dentre os efeitos colaterais mais comuns, estão a comunicação com o servidor, tanto para carregamento quanto para escrita dos dados, e comunicação com outros componentes internos do *frontend*, como o combinador de turmas (bloco (4)) e o *Service Worker* (bloco (5)).

Essa comunicação com componentes internos, inclusive, acontece através do uso de mensagens, visto que tais componentes rodam em instâncias diferentes do motor Javascript no qual a página está sendo executada. Isso é feito tanto por questão da própria API do navegador, como no caso do *Service Worker*, onde apenas um *Service Worker* atende todas as instâncias da página aberta, como por questões de otimização, como no caso do combinador de turmas, que é executado em uma *thread* separada dentro do navegador para permitir a execução de tarefas pesadas computacionalmente (como encontrar combinações de turmas válidas, ou seja, sem conflitos e respeitando os filtros impostos pelo usuário, num conjunto com dezenas de milhares de combinações) sem afetar a experiência do usuário com a interface.

4.3.6 *Service Workers*

O *service worker* é uma espécie de *proxy* que reside entre a página que o usuário abriu e o servidor, para prover suporte a navegação *offline* do sistema. Dentre os demais componentes listados, é o único que não é crítico, ou seja, caso não seja carregado ou suportado, não tende a causar problemas para o uso **normal** da página, uma vez que tem atuação transparente sob o *Frontend*, fazendo com que apenas o recurso de navegação *offline* se torne indisponível.

[11] <http://redux-saga.js.org>

O *service worker* trabalha armazenando e lendo dados salvos em um banco de dados disponibilizado por navegadores modernos acessível por uma API chamado *IndexedDB*. A implementação do *service worker*, conforme estrutura mostrada na Figura 31, é baseado na implementação de duas frentes de manipulação de dados: uma responsável por requisitar e armazenar os dados para navegação *offline* (bloco (6)), e outra para retornar tais dados da mesma forma que API *GraphQL* do *backend* (bloco (5)) faz (inclusive usando-se do mesmo modelo *GraphQL*), através do uso da implementação oficial do *GraphQL*, que permite que o *service worker* seja basicamente invísivel para o código executando na página do usuário, e intercepte as requisições sempre que possível.

Além do armazenamento dos dados através da API *IndexedDB*, também são usados APIs disponíveis especificamente para *service workers* que permitem que os recursos estáticos da página, como código Javascript, *Hyper Text Markup Language* (HTML) e CSS, sejam armazenados de maneira persistente no *cache* do navegador, e requisitados de maneira apropriada, quase que totalmente gerenciada pelo próprio navegador, conforme representado no bloco (8) da Figura 31.

Deste modo, a sincronização dos dados é feita em duas partes distintas, uma para o *cache* dos recursos da página, que é gerenciado automaticamente pelo navegador ao carregar o *service worker* (usando o *cache* de recursos estáticos representados no bloco (8) da Figura 31), e outra, manual, em que é feita a atualização dos dados até então armazenados no *IndexedDB*, usando uma API do *backend* desenvolvida especificamente para sincronização de dados, representado pelo bloco (6).

Esta sincronização manual dos dados foi desenvolvida de forma a transferir a menor quantidade de dados possível com o servidor, mas ainda sendo leve tanto para o cliente

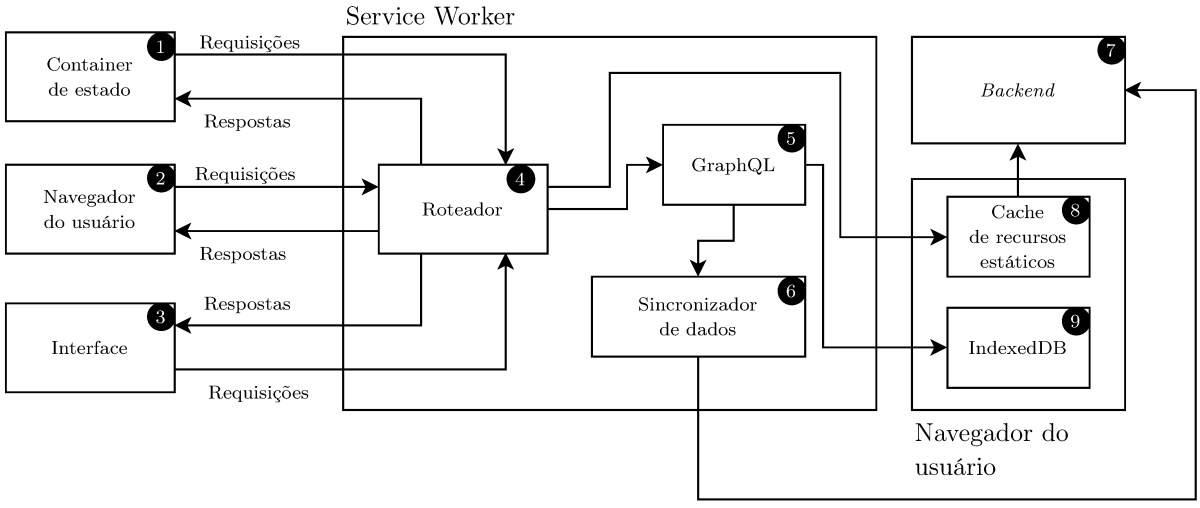

Figura 31 – Estrutura geral do *Service Worker*.

quanto para o servidor. Para conseguir esse feito, o cliente envia uma lista composta por identificador e versão de cada tipo de entidade que está armazenado atualmente no banco de dados local do navegador, junto de dados básicos como universidade, período e curso de interesse, que devem ser manualmente escolhidos pelo usuário afim de reduzir tanto o espaço necessário no dispositivo para armazenamento dos dados quanto para reduzir a carga no servidor. Com base nesses dados, o servidor detecta e retorna uma lista composta por dados a inserir ou atualizar, e também uma lista composta por identificadores a serem deletados da base. Desta forma, o *service worker* consegue rapidamente detectar o conjunto mínimo de operações a serem feitas para ter a base de dados atualizada, maximizando a performance da operação.

Por fim, para auxiliar e facilitar algumas tarefas rotineiras (como inicialização e uso do recurso de *cache* de arquivos estáticos do navegador, por exemplo) relacionadas à *service workers*, além de prover determinados recursos como o roteador (representado no bloco (4) da Figura 31), foi utilizado a biblioteca *Workbox*[12], desenvolvida pelo Google. É esse roteador que define tanto o que o *container de estado* (bloco (1)), o próprio navegador do usuário (bloco (2)) e a interface (bloco (3)) receberão como resposta ao fazer requisições para o servidor.

[12] <https://developers.google.com/web/tools/workbox>

5 Estudos de Caso

Os dados armazenados e exibidos hoje no sistema são coletados a partir de duas universidades: UFSC e USP. Essas universidades foram escolhidas especialmente por publicarem os dados necessários na internet, de forma acessível à mecanismos automatizados para extração de dados. Entretanto, por usarem abordagens diferentes, foi necessário o desenvolvimento de mais de um mecanismo para realizar a coleta de dados e a exportação desses dados em um formato comum para importação pelo sistema, em cada um dos formatos necessários de acordo com o tipo de dado a ser extraído da universidade.

Desta forma, os dados de ofertas possuem um formato comum, em JSON, enquanto que os dados de cursos possuem outro formato comum, também em JSON. Esses formatos precisam ser exportados por cada universidade individualmente, ou extraídos por terceiros e expostos nos formatos necessários. A estrutura de funcionamento para uma universidade está representada na Figura 32, sendo que os mecanismos representados pelos blocos (1) e (2) são totalmente independentes do mecanismo representado pelo bloco (3) em sua implementação, e se comunicam com o esse último mecanismo apenas a partir do formato JSON comum (que é diferente entre dados de ofertas e dados de cursos).

5.1 UFSC

Os dados da UFSC são coletados a partir de duas fontes principais, disponibilizadas à partir do Sistema de Controle Acadêmico da Graduação (CAGR): o cadastro de turmas - de onde são coletados dados de oferta de turmas a cada semestre - e os curriculos dos cursos fornecidos no formato PDF, de onde são coletados dados detalhados sobre as disciplinas, pré-requisitos e suas características em cada curso disponível na universidade. Esses dados são processados por dois mecanismos totalmente independentes, que são capazes de lidar com as peculiaridades de cada formato, e exportam, cada um, um arquivo JSON que contem todos os dados relativos à informação solicitada, em um formato padronizado e aceito pelo sistema principal de importação.

5.1.1 Captura de dados de ofertas

O mecanismo de captura de ofertas de disciplinas foi desenvolvido em Go e funciona sob demanda e de forma majoritariamente distribuída, sendo capaz de capturar dados do cadastro de turmas do CAGR disponibilizado pela UFSC. Como esse cadastro de turmas depende do uso de *HTTP cookies* para efetuar algumas funções (como paginação, por exemplo) corretamente, o sistema persiste essa informação entre cada acesso feito para

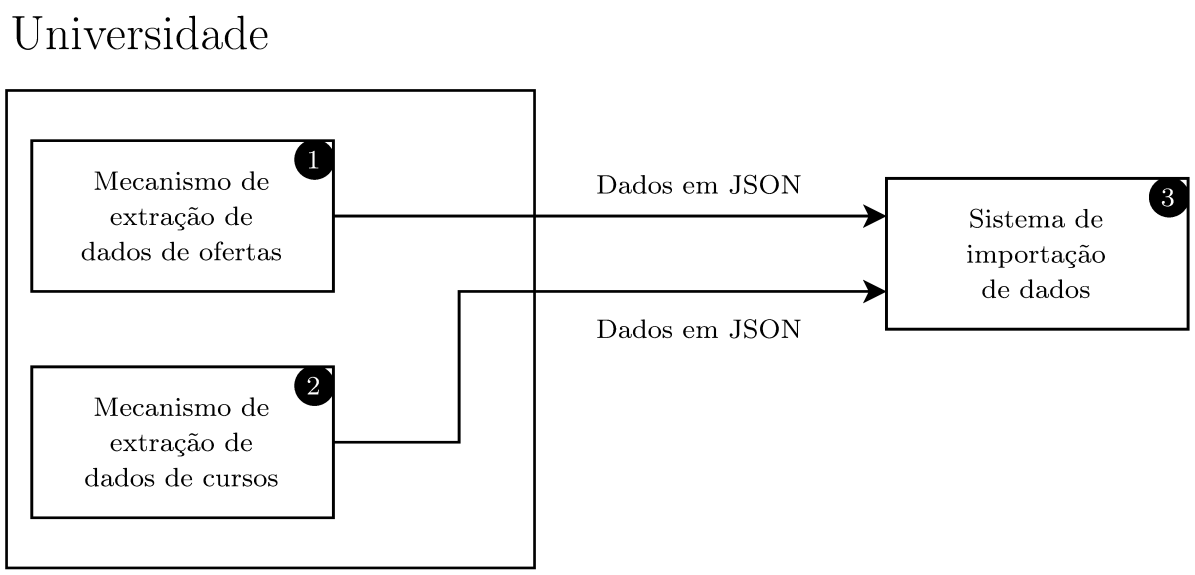

Figura 32 – Estrutura interna da comunicação entre uma universidade e o sistema de importação de dados.

cada conjunto de semestres e campus, e realiza tais acessos de maneira sequencial.

Dito isso, a primeira tarefa que o mecanismo realiza é acessar o cadastro de turmas da UFSC (bloco (1) da Figura 33) e identificar quais os semestres e campus disponíveis no sistema da universidade (bloco (2)). A partir daí, são disparadas tarefas, executadas paralelamente, para cada conjunto de semestre e campus (bloco (3)), e cada tarefa coleta os dados para esse conjunto de maneira sequencial, cada uma gerando um arquivo JSON. Quando todas as tarefas disparadas são terminadas, uma outra tarefa é agendada para concatenar todos os arquivos gerados para cada combinação de semestre e campus (bloco (4)). O arquivo JSON gerado pode, então, ser enviado para o sistema de importação de dados, representado no bloco (5).

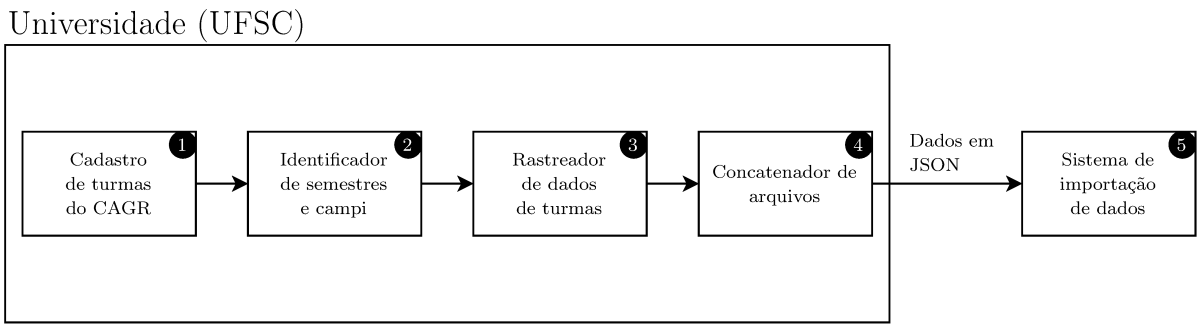

Figura 33 – Estrutura interna do mecanismo de captura de dados de ofertas da UFSC.

Desta forma, ao término da execução, apenas um arquivo JSON é gerado, contendo todos os dados dos últimos semestres conforme configurado pelo usuário manualmente e seguindo o formato necessário para o processamento pelo sistema de importação de dados do sistema. O nome desse arquivo é, então, disponibilizado em local público e enviado a esse sistema de importação, que então realiza a importação dos dados capturados para o banco de dados. Note que, devido a essa manipulação dos arquivos JSON, é necessário que todas as unidades de execução de tarefas deste mecanismo possuam acesso a um mesmo armazenamento compartilhado, de forma que os arquivos possam ser lidos apropriadamente pelo concatenador de arquivos.

Dado a natureza do sistema da UFSC, foram adicionados algumas proteções contra problemas comuns ao capturar dados. Uma dessas proteções é a detecção de páginas duplicadas do cadastro de turmas, que ocorre através do armazenamento do *hash* de cada página retornada pelo sistema da UFSC, o que permite detectar erros relacionados à perda da sessão do sistema. Além disso, o Algoritmo é capaz de iniciar uma nova sessão automaticamente, de forma que a captura dos dados possam continuar caso este erro ocorra.

5.1.2 Captura de dados de cursos

Já o mecanismo de leitura dos currículos em formato *Portable Document Format* (PDF) não faz nenhuma requisição externa ao site da UFSC. Tal mecanismo é, na verdade, um programa, representado no bloco (2) da Figura 34 e desenvolvido em Java, que usa a biblioteca **Apache PDFBox** para ler o PDF de cada currículo (a partir de uma lista fornecida no bloco (1)), identificar os cursos, habilitações, disciplinas e demais dados dentro do arquivo e exportar, então um arquivo JSON contendo todas as informações identificadas no arquivo em questão, em um formato aceito pelo sistema de importação de dados do sistema (bloco (3)). Dentre as informações capturadas, se encontram:

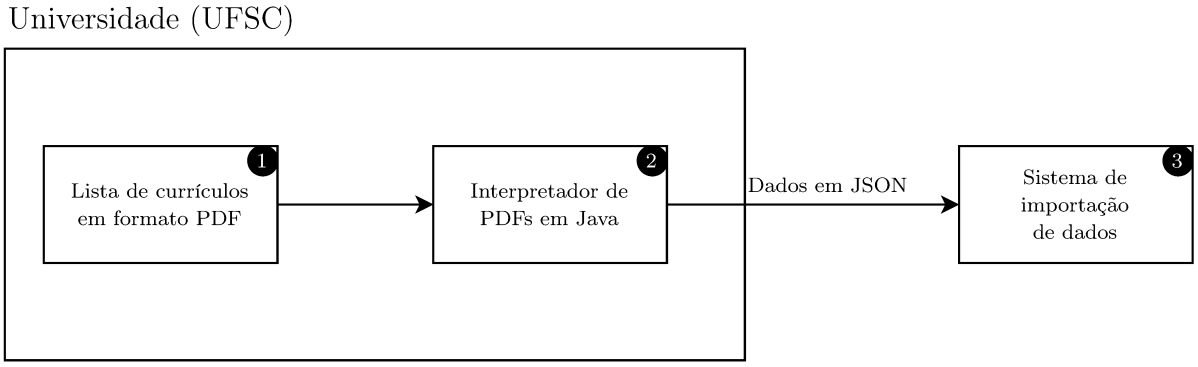

Figura 34 – Estrutura interna do mecanismo de captura de dados de cursos da UFSC.

- **Curso**: Código, Nome, Habilitações;
- **Habilitação**: Código, Nome, Fases;
- **Fase**: Nome, Disciplinas;
- **Disciplina**: ID,Código, Nome, Tipo, Descrição, Disciplinas relacionadas;

Esse sistema não é distribuído e foi desenvolvido desta forma devido ao fato de que esses dados não são atualizados com frequência, e a coleta de dados a partir de arquivos PDF é algo que ainda não foi implementado consistentemente em Go. Portanto, foi escolhido Java para esta tarefa especialmente devido à implementação do Apache PDFBox, que permitiu que tal implementação fosse possível.

Mesmo sendo desenvolvido em outra linguagem de programação, o fato do resultado do programa ser uma saída padronizada em JSON permite que o resto do sistema, desenvolvido em Go, consiga ler e importar os dados de maneira apropriada, permitindo aproveitar uma das vantagens em se ter um formato comum na comunicação entre sistemas desenvolvidos em diferentes linguagens.

5.2 USP

Os dados da USP são coletados a partir do Jupiter Web, a partir de duas implementações distribuídas em Go que acessam as páginas necessárias dentro do sistema do Jupiter Web (que é o sistema de graduação da USP) e identificam, a partir das páginas públicas desse sistema, todos os dados desejados por cada implementação, ou seja, os dados de ofertas de turmas para o semestre atual e também os dados relacionados ao currículo de cada curso que a universidade oferece. Ao final do processo, cada implementação gera um arquivo JSON único contendo os dados identificados no formato necessário para importação pelo robô do sistema.

As principais diferenças entre a UFSC e a USP no aspecto de coleta de dados ao fato de que a USP não depende de *HTTP Cookies* ou armazenamento de sessão para acesso aos dados, e, na USP, todos os dados são fornecidos em HTML, o que facilita consideravelmente a implementação do robô e evita a necessidade do uso de um programa em Java para realizar o processamento. Por outro lado, a quantidade de dados encontrado na USP é muito maior, dado que a universidade fornece, aos seus alunos, muito mais cursos e disciplinas do que a UFSC hoje fornece, o que faz com que a execução dos mecnaismos, mesmo que quase totalmente distribuída e paralela, seja mais demorada do que os mesmos mecanismos de coleta de dados para a UFSC.

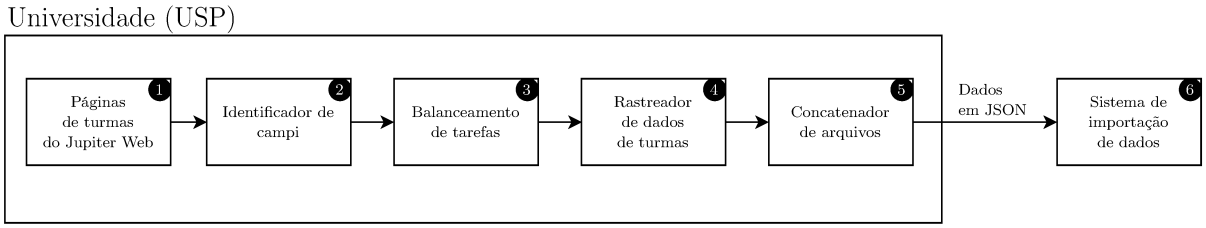

Figura 35 – Estrutura interna do mecanismo de captura de dados de ofertas da USP.

5.2.1 Captura de dados de ofertas

O mecanismo de captura de ofertas de disciplinas da USP foi desenvolvido em Go e funciona, assim como o mecanismo de captura de dados de ofertas da UFSC, sob demanda e de forma distribuída. Pelo fato do Jupiter Web não depender de *HTTP Cookies*, uma grande parte do algoritmo é executada de maneira paralela, com apenas o balanceamento de tarefas - representado no bloco (3) da Figura 35 - e o concatenador de arquivos (bloco (5)) sendo etapas que não podem ser paralelizadas devido à seu funcionamento: no caso do balanceamento de tarefas, é necessário capturar a lista de campus e disciplinas a serem rastreadas (bloco (2)), a partir das páginas de turmas do Júpiter Web (bloco (1)), e então dividir igualmente o numero de tarefas a serem executadas de acordo com o numero de unidades de execução paralelas definido, enquanto que no caso do concatenador de arquivos todos os arquivos devem ser lidos, um a um, e seus valores escritos em um arquivo de saída, de maneira sequencial.

Após tal balanceamento de tarefas, são disparadas tarefas que rastreiam, a partir do *parsing* do HTML da página, os dados de turmas para cada disciplina detectada, gerando arquivos JSON com os dados já coletados até então que contem um subconjunto das ofertas de turmas disponíbilizadas pela USP, correspondente às disciplinas acessadas. Isso é representado no bloco (4) da Figura 35.

Após a captura dos dados de ofertas e o término da execução de todas as unidades de execução paralelas, o concatenador de arquivos, representado no bloco (5) da Figura 35, é então executado, e recebe uma lista com os arquivos gerados e lê cada arquivo e passa os dados, ainda em formato JSON, para um único, novo, arquivo, que contem todos os dados capturados pelo mecanismo, e que é a saída final do mecanismo, no formato comum para ofertas aceito pelo robô de importação de dados, representado no bloco (6).

5.2.2 Captura de dados de cursos

O mecanismo de captura de dados de cursos da USP é desenvolvido em Go e é, também, majoritariamente paralelo, com apenas o balanceamento de tarefas e o concatenador de arquivos sendo etapas que não podem ser paralelizadas devido à seu

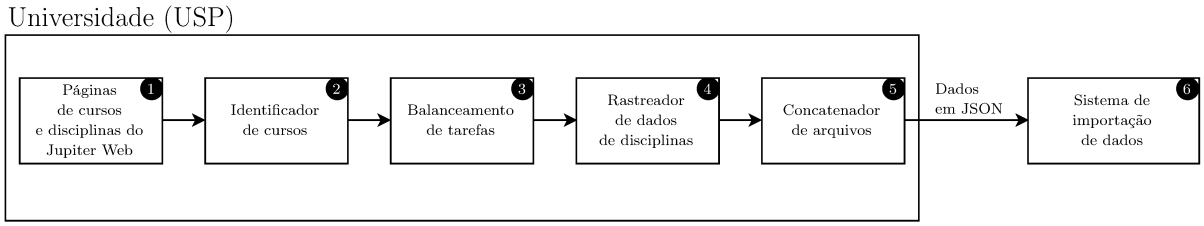

Figura 36 – Estrutura interna do mecanismo de captura de dados de cursos da USP.

funcionamento, conforme explicado na seção 5.2.1.

A captura dos dados de cursos acontece de forma parecida com a captura de dados de ofertas: o mecanismo realiza a identificação da lista de cursos (bloco (2) da Figura 36) a partir dos dados de cursos e disciplinas fornecidos pelo Jupiter Web ((1)), faz a divisão das tarefas pela quantidade de tarefas a serem executadas paralelamente (bloco (3)), rastreia os dados a partir do *parsing* do HTML das páginas e emite arquivos JSON contendo esses dados (bloco (4)), que são então concatenados por uma tarefa executada quando todo o rastreamento já foi concluído (bloco (5)), que gera, então, um único arquivo JSON contendo todos os dados capturados, que pode ser enviado para o sistema de importação de dados (bloco (6)).

Uma característica importante deste mecanismo é que a captura de detalhes (tais como descrição da disciplina, por exemplo) das disciplinas é **opcional**. Isso acontece pois, ao fazer a captura desses dados, o processo de rastreamento se torna muito mais lento, visto que o mecanismo precisa acessar a página de cada disciplina individualmente para obter tais informações, e tal procedimento não pode ser balanceado para processamento paralelo devido ao fato desses dados estarem diretamente associados com cada curso , com o processo de balanceamento inicialmente realizado não sendo tão eficiente nesse caso.

6 Experimentos

Durante o desenvolvimento das diferentes partes do sistema, ao realizar testes de funcionamento, foi possível verificar diferentes dados que podem ser analisados para permitir, a usuários que não participaram do desenvolvimento do sistema, uma breve noção dos diferentes desafios envolvidos durante esse processo. Muitos desses dados são, de fato, "invísiveis" para o usuário final, que vai usar o sistema para simular sua matrícula, entretanto, mesmo esses dados possuem importância fundamental na viabilidade do mecanismo.

Um exemplo básico de dado desse tipo é o consumo de memória RAM durante a importação de dados: computadores não possuem memória ilimitada, portanto, é desejado que o sistema possua baixo consumo de memória de forma a possibilitar tanto a manutenção da *performance* do mecanismo para todos como também permitir que mesmo universidades tão grandes quanto a USP, por exemplo, possam ter seus dados importados e fornecidos dentro do mecanismo sem exigir um computador com muita memória RAM, e portanto mais caro, para que isso seja possível.

Entretanto, existem dados que afetam diretamente o usuário. Um exemplo disso é o tamanho dos recursos que compõem o *frontend*: tais recursos são enviados diretamente para o navegador do usuário e, mesmo que possam ser compactados (reduzindo a quantidade de dados consumido na franquia móvel, por exemplo), ainda precisam ser executados no dispositivo local, do usuário, o que implica numa possível lentidão, e até travamentos caso o dispositivo do usuário não tenha capacidade de processamento suficiente para lidar com o código da página.

Por isso, a análise de dados, envolvendo diferentes experimentos, envolverá três partes principais: uma para os coletores de dados, apresentados no Capítulo 5, uma para o *backend*, apresentado na Seção 4.2, e, por fim, uma para o *frontend*, apresentado na Seção 4.3.

Um aspecto importante a ser considerado em relação aos testes de uso de memória apresentados é que esses testes considerarão, apenas, o *Unique Set Size* (USS), que é a quantidade de memória que seria liberado se o processo fosse terminado nesse exato instante. Isso é feito pois, dependendo de fatores como o banco de dados (explicado na Seção 4.2.4.1) usado, por exemplo, o uso de memória do aplicativo pode ser maior em decorrência do uso de bibliotecas compartilhadas com outros aplicativos, e também com o mapeamento de arquivos em memória (cuja quantidade de dados em memória é administrado completamente pelo sistema operacional).

Todos os testes executarão em um computador com Ryzen 7 1700, 16GB de RAM e

SSD 1TB NVME, mas, mesmo com essa configuração, todos os testes foram feitos usando apenas 4 unidades de execução, visando reduzir memória e mostrar condições "ideais" de uso. Além disso, o banco de dados usado é o *Bolt*, sem *cache*, e com o uso do codificador *Gob* sem compressão para codificação dos dados. Por fim, o programa foi compilado usando *Go 1.13* e executado em sistema operacional Antergos Linux.

6.1 Coletores de dados

Os coletores de dados desenvolvidos para a UFSC e para a USP são ferramentas que a princípio não precisam rodar no mesmo servidor que o resto do sistema, mas que lidam com uma enorme quantidade de informações ao depender, basicamente, do *parsing* de páginas HTML fornecidas pelas universidades. Isso acontece pois tais sistemas precisam acessar uma quantidade muito maior de páginas, e interpretar muito mais conteúdo, para poder extrair apenas os dados que interessam.

Devido a isso, é necessário que esses coletores sejam otimizados para consumir pouca memória mesmo ao acessar muitas páginas. Isso é feito principalmente com o uso de técnicas de *parsing* iterativo, que permitem ir descartando todo o conteúdo da página que não é interessante para extração a partir do momento em que esse conteúdo é encontrado. Além disso, mesmo o conteúdo já extraído não é armazenado em grandes quantidades na memória, mas imediatamente codificado e transferido para o armazenamento persistente, o que facilita tanto na recuperação de erros quanto na diminuição do consumo de memória.

6.1.1 UFSC

No caso da UFSC, a extração de dados mais complexa acontece na extração de dados de cursos, que precisam lidar com o formato PDF, que acaba exigindo o uso de uma biblioteca Java para fazer a interpretação do conteúdo, e a construção de grafos complexos para cada curso, com a necessidade de armazenar os dados de cada disciplina em memória durante o processamento do curso, isso acaba exigindo maior consumo de memória, e, devido ao *parsing* do PDF, também de processador.

Em relação ao extrator de dados de turmas da UFSC, o maior desafio é o armazenamento da sessão (salvos em HTTP *cookies*), mas isso não implica em maior consumo de memória nem processamento visto que a quantidade de dados que é necessário armazenar é pequena. Além disso, como não há a necessidade de criar um grafo baseado na relação entre as disciplinas, é possível detectar e persistir os dados de turma coletados diretamente para o disco, reduzindo o consumo de memória.

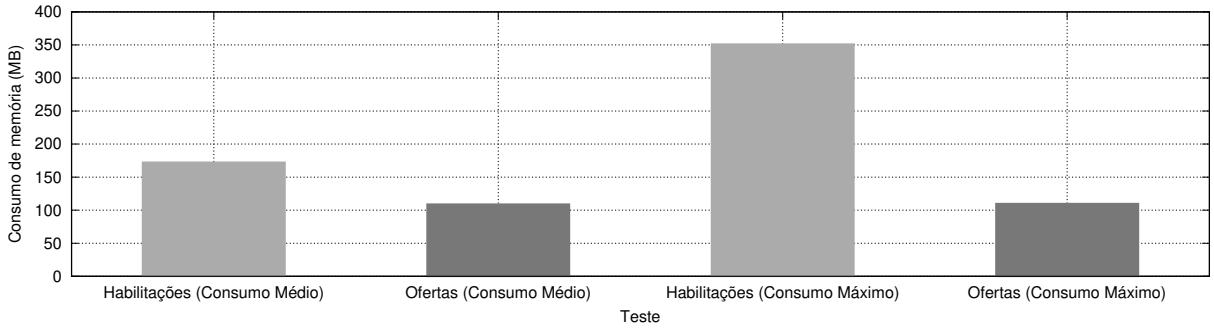

Figura 37 – Comparação de consumo de memória entre os coletores de dados da UFSC.

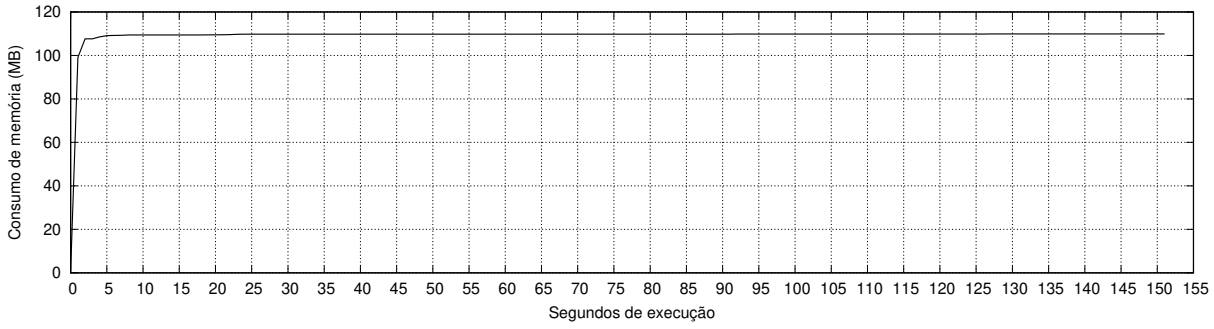

Figura 38 – Registro de consumo de memória durante a execução do coletor de dados de turmas dos últimos 3 semestres da UFSC.

6.1.1.1 Uso de memória

O uso de memória dos coletores da UFSC, como já antecipado, foi maior no extrator de dados de cursos, devido ao fato de ser desenvolvido em Java e precisar lidar com o *parsing* de arquivos PDF, incluindo o armazenamento temporário de todas as disciplinas de curso e suas devidas relações. É possível observar esas diferença no consumo de memória entre os coletores de dados da UFSC na Figura 37.

Um aspecto importante que vale destacar nessa comparação é que, a princípio, o coletor de dados de habilitações compreende apenas a parte de interpretação dos arquivos PDF, sem, portanto, considerar *download* e rastreamento desses arquivos.

Por fim, foi possível observar, através da Figura 38, que o consumo de memória do coletor de dados de turmas da UFSC possui uso de memória razoavelmente constante, enquanto que o coletor de dados de cursos/habilitações quase que continuamente aumenta o consumo de memória durante a execução, como é possível visualizar na Figura 39, com quedas em alguns momentos devido ao próprio *Garbage Collector* do Java.

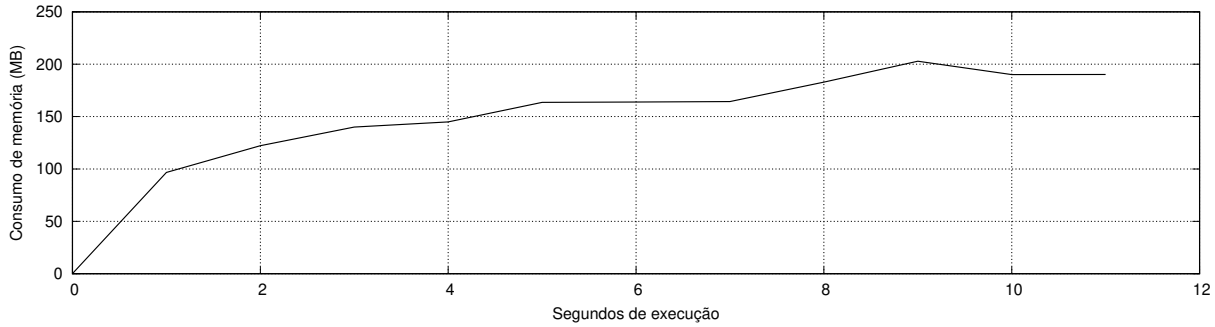

Figura 39 – Registro de consumo de memória durante a execução do coletor de dados de cursos/habilitações da UFSC.

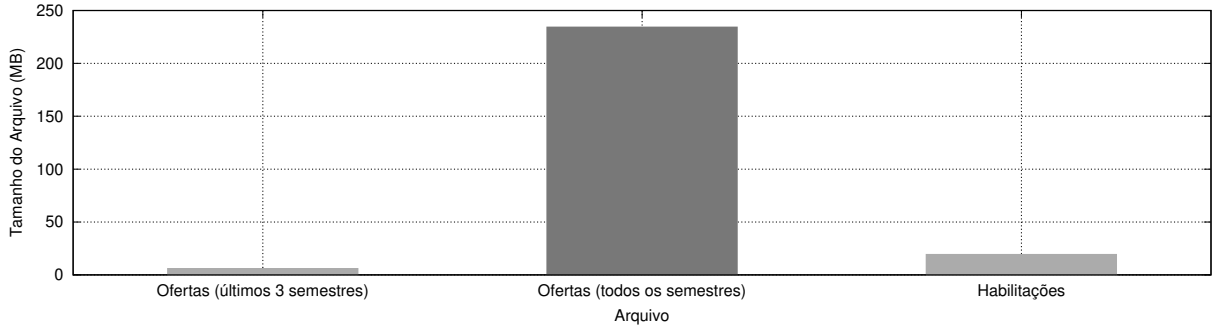

Figura 40 – Tamanho dos arquivos gerados durante a execução dos coletores de dados da UFSC.

6.1.1.2 Tamanho dos arquivos

Em relação ao tamanho dos arquivos gerados com os dados da UFSC, mostrados na Figura 40, foi possível observar que, salvo nos casos onde os dados de turmas de todos os semestres da universidade foram coletados, o tamanho do arquivo gerado pelo extrador de cursos foi sempre maior que os dados de turmas dos semestres recentes coletados. Isso acontece pois a UFSC tem atualmente cerca de 127 cursos, distribuidos por 279 habilitações, com um total de 19806 disciplinas, mas a maior parte dessas não possui ofertas semestralmente: Em 2019-2, por exemplo, a UFSC ofereceu 6613 turmas, distribuídas por apenas 3741 disciplinas.

Isso faz com que, ao coletar 3 semestres de dados de ofertas de turmas (o que corresponde a 1 ano para a UFSC), o arquivo com esses dados fique com menos da metade do tamanho do arquivo com os dados de cursos, como é possível visualizar na Figura 40.

Por fim, como é possível reparar na Figura, os dados podem ser compactados e, ao fazer isso, o tamanho dos arquivos diminui drasticamente, como mostrado na Figura 41, ajudando a reduzir a necessidade de um grande espaço de armazenamento para os dados,

além de não perder a compatibilidade com o resto do sistema de importação (caso a ferramenta de compactação seja feita em formato *GZIP*, que é suportado nativamente desde que configurado corretamente).

6.1.1.3 Duração da coleta de dados

No caso da duração para coleta dos dados da UFSC, foi observado que o coletor de dados de habilitações/cursos é muito mais rápido que o coletor de dados de turmas, como mostrado na Figura 42. Isso acontece pois o coletor de dados de turmas, além de fazer o *parsing* e extração dos dados a partir das páginas do CAGR, também precisa acessar cada página individualmente, enquanto que o coletor de dados de habilitações/cursos apenas faz o *parsing* a partir dos arquivos PDF locais dos currículos de cursos da UFSC, que o usuário deve baixar por conta própria.

Essa diferença de abordagem já é o suficiente pra causar tal vantagem nos tempos de execução em comparação com a coleta de dados de turmas da UFSC, uma vez que, embora o download das páginas por si só seja lento, está sujeito a limitações do próprio CAGR, como o fato de não poder fazer as requisições de maneira totalmente paralelizada, e também a necessidade de colocar um intervalo entre cada requisição para que o servidor da universidade não fique sobrecarregado com o excesso de conexões, levando ao eventual bloqueio de acesso ás páginas.

6.1.2 USP

Em relação à USP, o maior desafio que os coletores de dados enfrentam é lidar com a quantidade de dados que a universidade disponibiliza, especialmente quando comparado com a UFSC. Essa quantidade de dados leva a desafios tanto em termos de armazenamento, pois o arquivo final com os dados tende a ficar muito grande, quanto ao consumo de

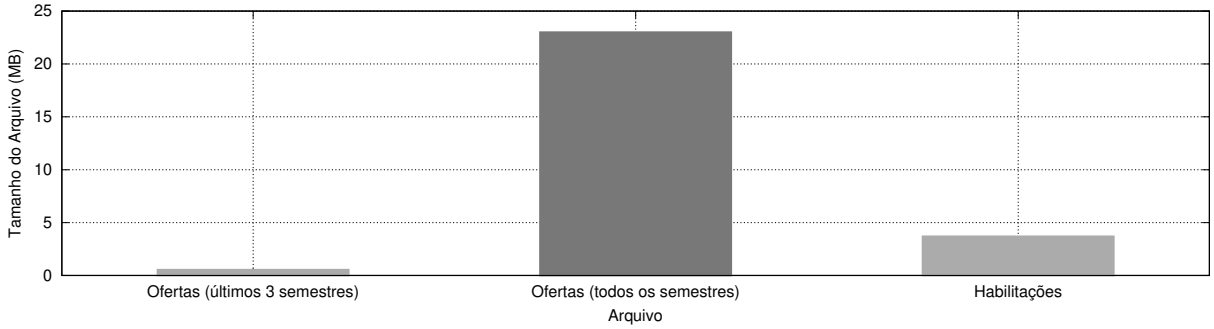

Figura 41 – Tamanho dos arquivos gerados durante a execução dos coletores de dados da UFSC, compactados usando *GZIP*.

memória, que pode crescer absurdamente caso o algoritmo não seja cuidadosamente planejado.

Outro aspecto que chama atenção, também, é o fato de que devido ao tamanho da universidade e a própria estrutura do site da USP, se torna necessário acessar uma quantidade enorme de paǵinas, o que faz com que a execução dos coletores de dados acabe ficando muito mais lenta caso o sistema não seja apropriadamente desenvolvido para executar de maneira paralela, dividindo, para várias unidades de execução, o trabalho de acessar e extrair dados das páginas. Assim como no caso da UFSC, no entanto, é necessário limitar a velocidade com que as páginas são acessadas, de forma que não sobrecarregue os servidores da universidade e leve ao bloqueio das páginas com os dados para o público.

6.1.2.1 Uso de memória

Devido ao fato dos coletores de dados da USP serem deesenvolvidos em Go, foi possível observar um baixo consumo de memória, como mostrado na Figura 43, muito menor do que o coletor de dados de cursos da UFSC, que foi desenvolvido em Java.

Esse consumo da memória, assim como o coletor de dados de turmas da UFSC, é praticamente constante, com uma característica importante relacionada à própria linguagem se destacando em ambos os coletores de dados: O fato de que, após alguns minutos, ou até mesmo horas, de execução, o consumo de memória cai drasticamente, em decorrência da ação do *garbage collector* da linguagem, como é possível visualizar tanto na Figura 44 como na Figura 45.

Durante a maior parte da execução do programa, entretanto, foi possível observar um crescimento extremamente pequeno da quantidade de memória alocada. Isso se deve à priorização do Go, com o auxílio do próprio sistema (através do uso intensivo de reciclagem de memória), em reusar blocos de memória já alocados em vez de constantemente liberar e alocar novos blocos, o que possibilita maior desempenho do sistema. Em alguns casos, entretanto, foi possível observar quedas no consumo de memória apenas quando o algoritmo

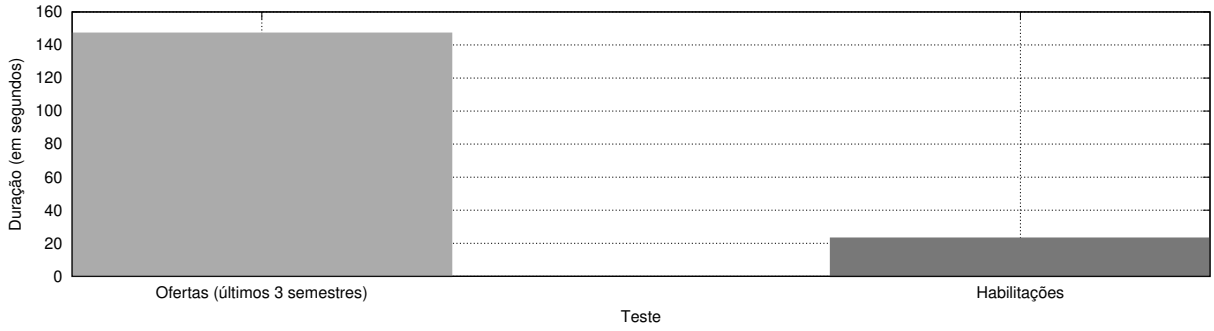

Figura 42 – Duração da execução das coletas de dados da UFSC.

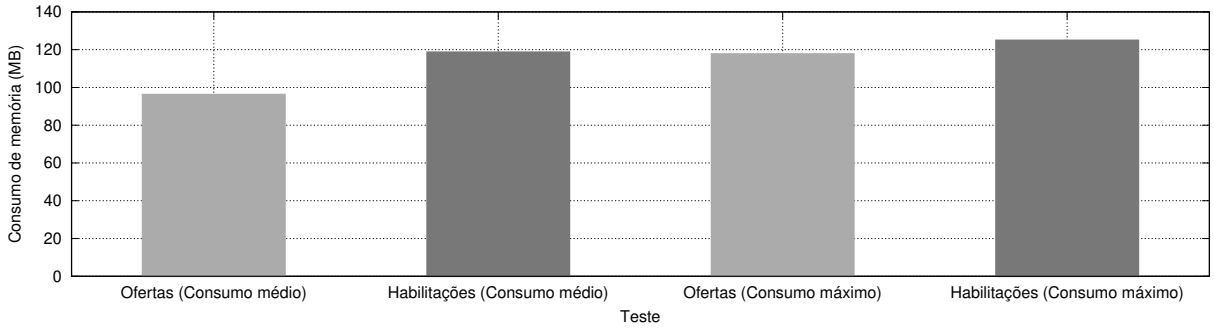

Figura 43 – Comparação de consumo de memória entre os coletores de dados da USP.

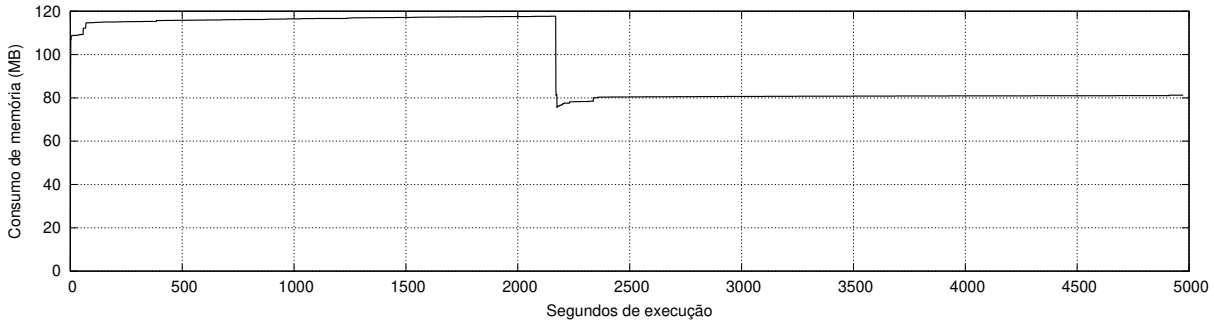

Figura 44 – Registro de consumo de memória durante a execução do coletor de dados de turmas da USP.

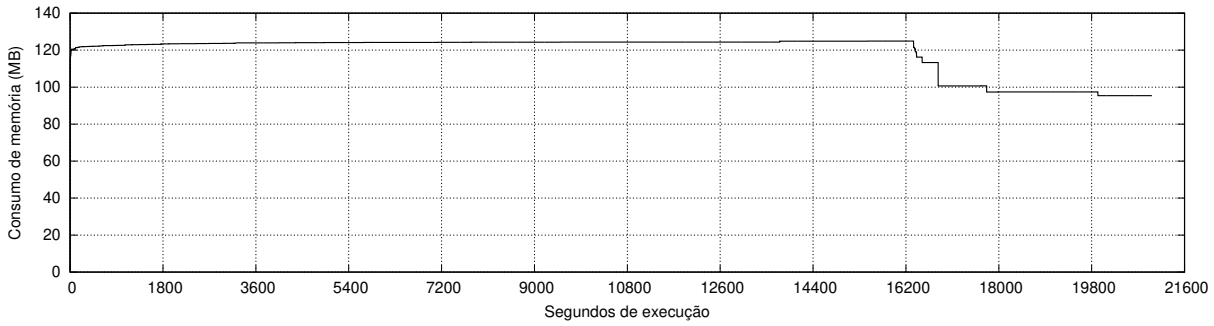

Figura 45 – Registro de consumo de memória durante a execução do coletor de dados de habilitações/cursos da USP.

já estava quase no fim de sua execução, devido à diminuição na carga de trabalho em decorrência de um balanceamento de tarefas não-perfeito, que faz com que determinadas unidades de execução fiquem sem tarefas para fazer e resulta na diminuição do uso da memória do programa, como mostrado na Figura 45.

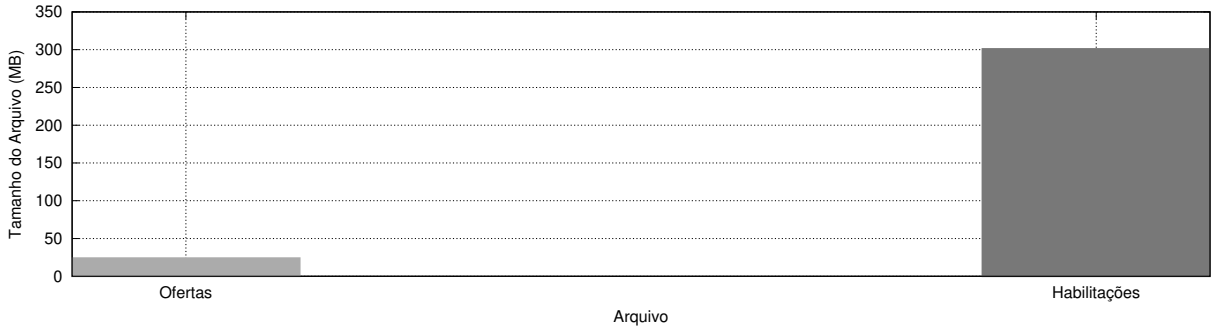

Figura 46 – Tamanho dos arquivos gerados durante a execução dos coletores de dados da USP.

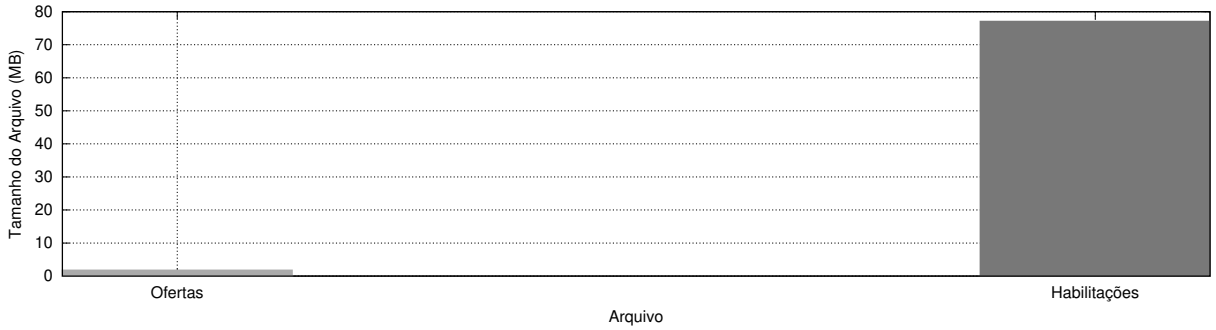

Figura 47 – Tamanho dos arquivos gerados durante a execução dos coletores de dados da USP, compactados usando *GZIP*.

6.1.2.2 Tamanho dos arquivos

Já no caso do tamanho dos arquivos gerados da USP, foi possível notar que o tamanho do arquivo com os dados de cursos é muito maior que o tamanho do arquivo de dados de turmas, como mostrado na Figura 46. Isso acontece por um motivo similar ao da UFSC, mas em uma escala muito maior, devido ao fato de que a USP fornece apenas o último semestre de dados de turmas (diferente da UFSC, onde é possível definir o número de semestres a ser capturado), e também devido ao fato de que a USP possui um número muito maior de cursos e de turmas: são 131 cursos, distribuídos por 765 habilitações e 57408 disciplinas. Em relação aos dados de ofertas, para um único semestre, foram detectados 11391 turmas distribuídas por 5638 disciplinas, o que acaba por justificar o tamanho maior do arquivo quando comparado com a UFSC, por exemplo.

Quando esses arquivos são compactados usando *GZIP*, conforme mostrado na Figura 47, a redução de espaço consumido em relação aos arquivos não-compactados é maior do que no caso da UFSC, devido ao fato de que, por possuir mais dados e, naturalmente, muitos desses dados acabarem sendo repetidos, é possível que o algoritmo

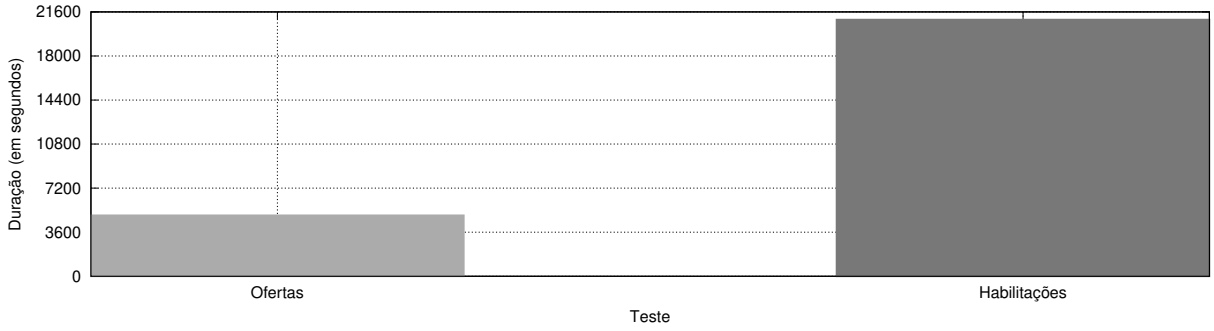

Figura 48 – Duração da execução das coletas de dados da USP.

de compressão possa compactar melhor os dados. Entretanto, mesmo com esse fator, foi possível observar que o arquivo contendo os dados de habilitações/cursos ainda ficou grande, ocupando 77MB mesmo após a compactação com *GZIP*.

6.1.2.3 Duração da coleta de dados

Em relação à duração da coleta de dados da USP, foi possível observar que, diferente da duração da coleta de dados da UFSC, o coletor de dados de habilitações/cursos demorou muito mais para executar do que o coletor de dados de ofertas, como mostrado na Figura 48. Isso se deve, principalmente, ao fato de que o coletor de dados de habilitações/cursos da USP precisa acessar uma quantidade enorme de páginas, afim de capturar os dados de todas as disciplinas que a universidade oferece, além do fato de que, diferente do caso da UFSC, a USP não fornece páginas que detalham todas as informações que o coletor de dados procura coletar, como, por exemplo, descrições da disciplina, o que faz com que seja necessário acessar uma página para cada disciplina afim de obter tais informações.

Devido a isso, o sistema precisa acessar, sequencialmente, as páginas de cada disciplina, para cada curso encontrado, o que aumenta e muito o tempo necessário para capturar todos os dados de um curso, por exemplo. Além disso, a USP também fornece dados de versões anteriores dos cursos, que também são coletados, o que acaba por aumentar muito o número de disciplinas encontrados e, portanto, o tempo necessário para coletar todos os dados.

Enquanto isso, no caso do coletor de dados de ofertas, o sistema acaba capturando apenas os dados do último semestre disponibilizados pela universidade, e consegue paralelizar muito bem o acesso às páginas para coleta de informação entre as diferentes unidades de execução, o que permite que o sistema como um todo seja mais rápido em relação à coleta de dados de habilitações/cursos, mas, devido ao tamanho da universidade, ainda seja mais lento do que a coleta de dados de ofertas da UFSC, por exemplo.

6.2 Backend

O *backend* do sistema é responsável por importar os dados das universidades e, além disso, atender as requisições dos usuários. Devido a isso, é a parte que precisa ser mais otimizada no sistema, pois precisa lidar com uma grande quantidade de dados, tanto em termos de processamento quanto em termos de gravação (durante o processo de importação) e leitura (durante as requisições do usuário).

O momento mais desafiador, nesse caso, é durante o processo de importação, uma vez que o sistema precisa ler os arquivos JSON (que podem ser grandes,a ponto de possuir mais de 100MB no caso da USP, como mostrado na seção anterior), fazer a comparação dos dados com o arquivo mais recentemente processado e então importar esse arquivo paralelamente, tanto diretamente para o banco de dados quanto para o índice. Como os arquivos são grandes, importar esses arquivos exigem que o algoritmo não carregue todo o arquivo em memória, caso contrário o consumo de memória cresceria de acordo com o tamanho dos arquivos, o que prejudicaria toda o desempenho do sistema de acordo com o tamanho do sistema.

Um detalhe importante sobre esse sistema é que as diferentes configurações de *cache* e banco de dados podem influenciar diretamente algumas dessas métricas, devido a características próprias desses subsistemas. Um exemplo disso é o fato de que o uso do banco de dados *Bolt*, por exemplo, pode aumentar bastante algumas métricas de consumo de memória devido ao uso de arquivos mapeados em memória, que acabam fazendo com que o aplicativo acabe "consumindo" mais memória que, na verdade, é automaticamente gerenciada pelo sistema operacional e na tentativa de otimizar ao máximo o acesso ao arquivo de banco de dados de acordo com a quantidade de memória disponível.

Além disso, é importante observar que todas as análises foram feitas com o banco de dados inicialmente limpo, com apenas um usuário e as duas universidades correspondentes cadastradas, mas sem a presença de nenhum dado de turma ou de curso, de forma a facilitar a reprodução dos experimentos e também permitir uma análise melhor sobre o impacto da importação de cada conjunto de dados.

6.2.1 Uso de memória durante a importação de dados

Em relação ao uso de memória do sistema de importação de dados, mostrado na Figura 49 e na Figura 50, foi possível observar que o consumo depende, em parte, ao tamanho do arquivo a ser importado e aos tipos de dados armazenados. Durante a análise, foi possível notar que os dados da USP consomem um pouco mais de memória, mas não foi possível visualizar um crescimento linear quando em comparação com os dados de consumo de memória registrados para importação dos dados da UFSC, mostrando que não há uma relação estrita entre tamanho do arquivo e consumo de memória.

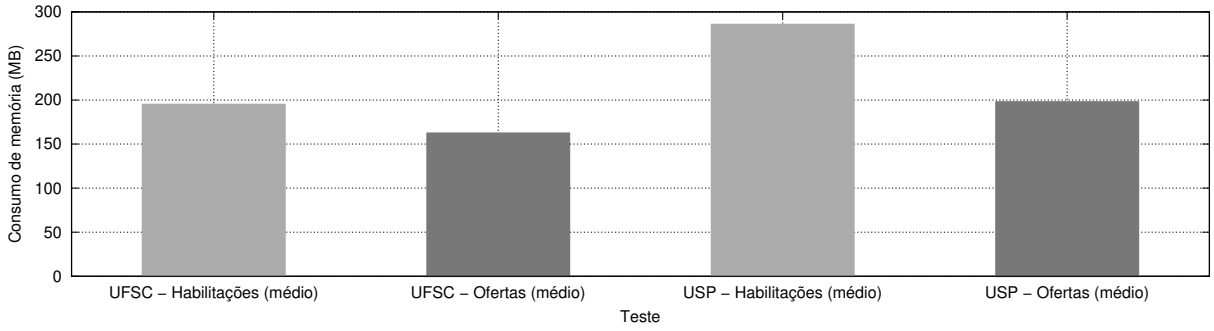

Figura 49 – Consumo de memória médio do sistema de importação de dados.

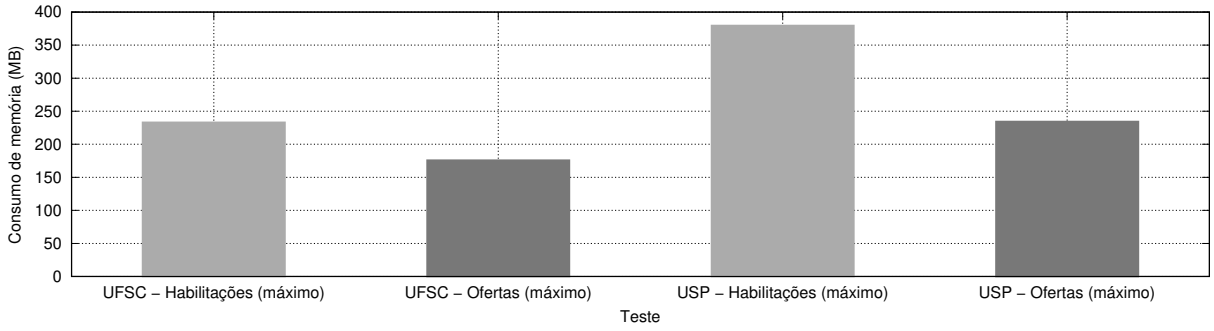

Figura 50 – Consumo de memória máximo do sistema de importação de dados.

Durante o processamento, foi possível observar um pico no consumo de memória de até 379MB, decorrente da presença de objetos grandes na memória e também ao fato de que o *runtime* Go sempre aloca memória adicional afim de evitar chamadas recorrentes ao sistema operacional. É possível observar os picos de memórias correspondentes na Figura 50.

Também foi possível observar, ao comparar o consumo médio com o consumo máximo, que, devido à longa duração da importação, no geral o consumo se mantem relativamente baixo, aumentando apenas em momentos mais críticos como a criação do índice de busca, que exige o armazenamento de mais dados temporariamente em memória para permitir a criação do índice reverso de modo otimizado.

6.2.2 Duração da importação

Sobre a duração de importação, apresentado na Figura 51, assim como no caso do consumo de memória da importação, foi possível observar que o tempo cresce de acordo com o tamanho do arquivo a ser importado, uma vez que a quantidade de registros cresce de acordo com o tamanho do arquivo e, também, não é possível paralelizar infinitamente a carga de trabalho.

Entretanto, assim como acontece com o caso do consumo de memória, o crescimento observado não foi linear, ou seja, não existe uma relação direta entre o número de registros de uma universidade e a duração da importação.

Um aspecto importante sobre as medições relacionadas à duração é que, pelo fato do sistema como um todo ser um servidor desenvolvido com a ideia de não ter um final específico de execução, a medição de duração do processo de importação ocorreu numa tarefa de monitoramento executada periodicamente e que é capaz de determinar se a importação acabou ou não. Essa tarefa periódica é executada a cada 1 minuto e, para efeitos de medição, foi adicionada uma condição onde ao detectar o término da importação o sistema encerra todo o servidor após um intervalo de 10 segundos. Ou seja, a margem de erro dos dados coletados é de, no máximo, 1 minuto e 10 segundos, no pior caso, mas ainda corresponde adequadamente à duração do processo de importação em um ambiente real dado a própria totalmente assíncrona natureza do sistema.

6.2.3 Tamanho do banco de dados

Em relação ao tamanho do banco de dados, mostrado na Figura 52, foi possível notar uma relação entre o tamanho do arquivo de entrada e o tamanho final do banco de dados, que aumenta numa escala maior que no caso da memória e da duração da importação. Isso é natural e esperado, devido ao fato de que todos os dados do arquivo de entrada são salvos e indexados de maneira à permitir o acesso e atualização mais eficiente possível aos dados, o que implica no uso de um maior consumo de espaço em disco conforme a quantidade de dados aumenta.

Graças à isso, tanto o banco de dados de turmas quanto a de habilitações para os dados da UFSC acaba por ser muito menor que o banco de dados correspondente para a USP, visto que esta última possui uma quantidade muito maior de dados que a primeira, tanto em dados de turmas quanto de habilitações.

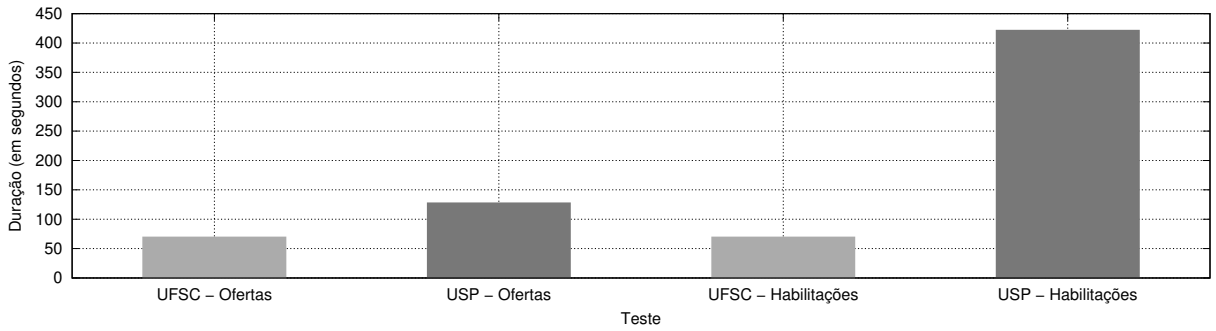

Figura 51 – Comparação da duração da importação de diferentes conjuntos de dados no sistema.

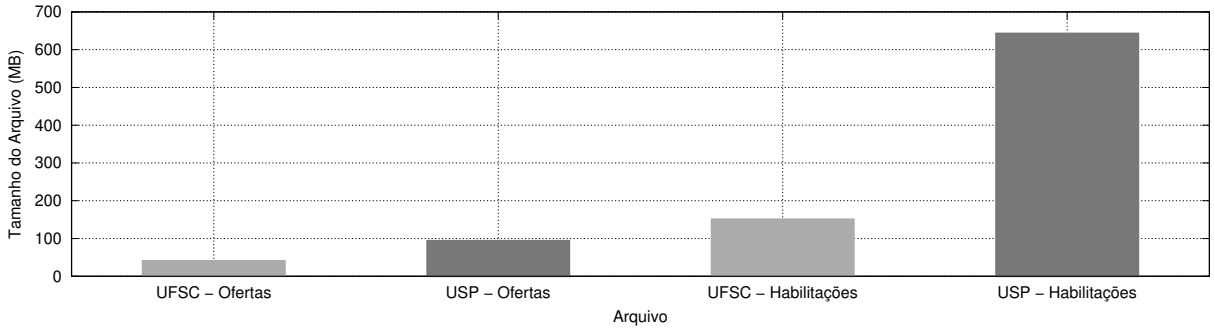

Figura 52 – Comparação do tamanho do banco de dados após a importação de diferentes conjuntos de dados no sistema.

Além disso, no banco de dados também são armazenados dados relacionados à execução das tarefas de importação, afim de permitir o gerenciamento das tarefas de maneira confiável. Isso significa que o tamanho do banco de dados acaba aumentando conforme a tarefa precisa de mais tarefas para sincronização da execução, mas essa parcela ainda é menor do que o espaço necessário para armazenamento dos outros dados, como as tabelas próprias com todos os dados e também os índices de busca.

6.3 Frontend

O *frontend* do sistema é a parte responsável por servir de interface entre o usuário e o resto do sistema, e que é executado diretamente no dispositivo do usuário. Graças a isso, é necessário que um cuidado especial na otimização e na quantidade de código a ser enviado para usuário para que a maior variedade possível de dispositivos seja suportado e, devido a esse, o sistema seja rápido de carregar e usar, mesmo em redes móveis de internet.

Devido a essas características, presentes em todo e qualquer *frontend*, surgiram diferentes metodologias de análise para determinar as propriedades dos sistemas e mensurar a experiência do usuário logo ao carregar a página, que é um dos momentos mais críticos, pois é quando há maior chance do usuário desistir de acessar a página: segundo um estudo [1] do Google, por exemplo, cerca de 53% dos usuários desistem de acessar uma página que demorou mais de três segundos para carregar.

Graças a isso, serão feitas três análises distintas em relação ao *frontend*: uma de tamanho dos arquivos, analisados individualmente e também no contexto de uma página de plano, uma análise com o *Lighthouse*, que foi desenvolvido justamente para analisar os diferentes aspectos que podem afetar a experiência do usuário em páginas *web*, como desempenho, acessibilidade e *Search Engine Optimization* (SEO), por exemplo.

[1] <https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/>

Nome	Tamanho
main.js	341KB
main.css	159KB
0.bundle.js	68KB
1.bundle.js	14KB
2.bundle.js	32KB
3.bundle.js	11KB
4.bundle.js	15KB
6.bundle.js	61KB
7.bundle.js	29KB
8.bundle.js	37KB
9.bundle.js	15KB
10.bundle.js	23KB
11.bundle.js	16KB
12.bundle.js	19KB
13.bundle.js	12KB
14.bundle.js	17KB
15.bundle.js	14KB
16.bundle.js	9.9KB
17.bundle.js	8.2KB
18.bundle.js	9.5KB
20.bundle.js	48KB
worker.js	7.7KB
sw.js	162KB

Tabela 2 – Nome e tamanho de cada arquivo gerado para o *frontend*. Com resultado já minimificado, mas sem compressão.

6.3.1 Tamanho dos arquivos

Em relação ao tamanho dos arquivos que compõem o *frontend*, foi possível observar a existência de arquivos relativamente grandes, devido principalmente ao uso de bibliotecas Javascript relativamente grandes por todo o site, como o *React*. Os dados de nome e tamanho do arquivo estão apresentados na Tabela 2.

Quem faz o gerenciamento em baixo nível dessa lista de arquivos é, a princípio, o *Webpack*[2], que recebe como entrada o arquivo com código fonte e aplica diferentes transformações para que esse código possa ser executado pelo navegador do usuário. Dentre essas transformações, está a detecção de recursos comuns para todos os arquivos, que são então colocados no *main.js* e carregados em todas as páginas. Neste caso, exemplos de recursos comuns seriam casos como bibliotecas, como *React* e *RMWC*, por exemplo, assim como componentes que são acessados em todas as páginas.

Já os outros arquivos são menores pois, a princípio, são arquivos projetados para

[2] <https://webpack.js.org/>

serem carregados sob demanda, quando o usuário navega por essas outras páginas, e possuem apenas os componentes, bibliotecas e demais recursos necessários por essas páginas, já ignorando tudo o que já está presente no arquivo principal. Desta forma, bibliotecas grandes como *React* acabam por ser carregadas apenas uma vez, o que ajuda a melhorar a eficiência do *frontend* como um todo e também diminui a quantidade de dados transferida entre o servidor e o dispositivo do usuário.

Um aspecto importante dos arquivos que constituem o *frontend* é que, devido ao fato dos *scripts* serem código que é executado no próprio navegador do usuário, tal código pode ser não só minimificado como também compactado usando algoritmos de compressão que são suportados nativamente pelo navegador, como *GZIP* e *Brotli*. O impacto desses algoritmos de compressão nos arquivos do *frontend* pode ser visualizado na Figura 53, que mostra o tamanho do maior arquivo do *frontend* (já minimificado) após a aplicação de cada algoritmo de compressão.

6.3.2 Análise com Lighthouse

A partir da análise do *frontend* feita com a ferramenta *Lighthouse*, foi possível reparar que, por sua própria natureza moderna, a aplicação já cumpre boa parte dos requisitos valiados pela ferramenta, como é possível reparar na Figura 54, com apenas alguns requisitos ainda não sendo cumpridos por falta de algumas informações extras, como ícones, por exemplo, que permitem que os navegadores (em especial, navegadores projetados para *smartphones*) tratem o *frontend* quase que como um aplicativo próprio.

Durante a execução da análise, que segundo a premissa do *Lighthouse* deve representar, de maneira aproximada, os aspectos de qualidade de um app da *Web*, foi possível observar que o sistema não avalia apenas aspectos próprios do *frontend*, mas também algumas configurações próprias do *backend*, como comunicação com o servidor através do protocolo HTTP 2.0 e também uso de conexão criptografada, por exemplo, que são

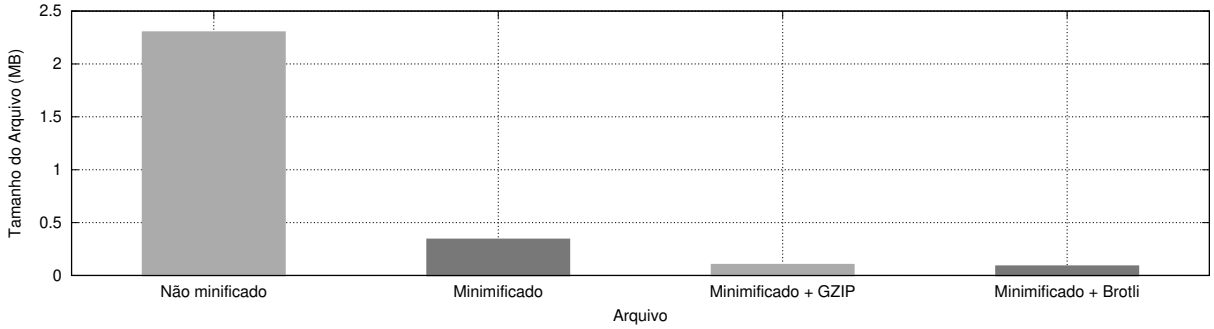

Figura 53 – Comparação do tamanho do maior arquivo do *frontend* ao aplicar técnicas para diminuir a quantidade de dados transferida entre o servidor e o cliente

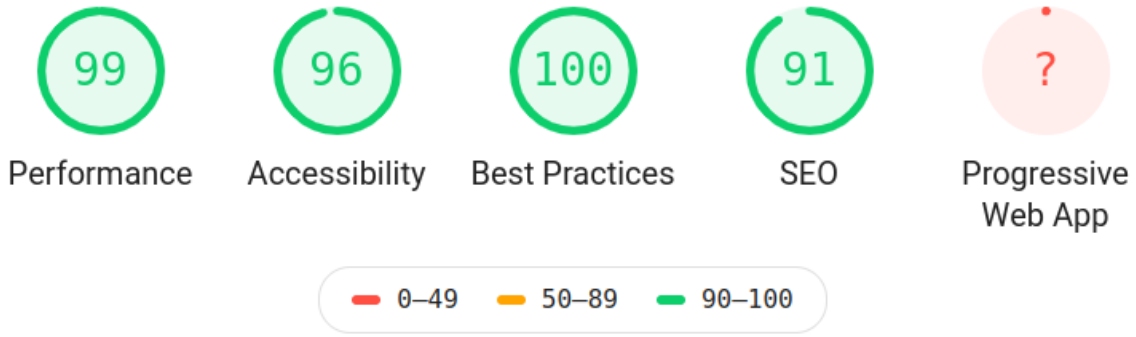

Figura 54 – Resultado da análise da página de plano com a ferramenta *Lighthouse*.

necessários para que alguns recursos do próprio *frontend* sejam liberados pelo próprio navegador, afim de garantir a própria segurança do usuário.

Por fim, por não ter chegado à nota máxima em alguns dos aspectos avaliados, o *Lighthouse* acabou deixando um conjunto de sugestões para melhoria na aplicação:

- Melhorar contraste entre cor da fonte e cor de fundo no cabeçalho;
- Pré-carregar determinados recursos, como fontes, por exemplo, antecipadamente;
- Configurar determinadas páginas para funcionarem *offline*;
- Adicionar descrição das páginas para reconhecimento pelos motores de busca;
- Configurar período de *cache* para os recursos estáticos da página.

Além disso, boa parte dos testes que o *Lighthouse* possui não podem ser executados de forma automática, e portanto não foram avaliados para a realização desta análise.

7 Conclusão

Neste trabalho, foi apresentado a proposta e o protótipo de um simulador de matrícula multi-institucional, com suporte a lidar com problemas complexos como validação de pré-requisitos e armazenamento de grande quantidade de dados sobre cursos e ofertas de turmas. Para lidar com tudo isso, foi desenvolvido uma estrutura flexível, altamente paralela e capaz de lidar com os diferentes aspectos das universidades de modo genérico, mas garantindo o acesso total à informação coletada. Com a proposta apresentada, é possível que os alunos criem planos para o próximo período letivo facilmente, tanto no computador quanto em um *smartphone*, para qualquer uma das universidades cadastradas no sistema (inicialmente, UFSC e USP).

Além disso, também é permitido que usuários externos cadastrem novas universidades dentro do sistema, algo que nenhum dos outros sistemas avaliados suporta. Dessa forma, além de fornecer, de imediato, um serviço aos alunos das universidades já cadastradas, é possível expandir ainda mais o suporte a outras universidades, sem a necessidade de desenvolvimento de mais sistemas complexos e isolados apenas para permitir esta tarefa.

Durante o desenvolvimento da plataforma, entretanto, algumas das ideias anteriormente consideradas acabaram não sendo implementadas, principalmente, devido à complexidade computacional envolvida. Um exemplo de funcionalidade assim é o sistema de notificações, que acabou sendo removido da implementação final por ser um problema díficil de lidar quando se opera um sistema onde uma mesma entidade pode ter milhares de seguidores, prontos para receber uma notificação.

Também ficou de fora a ideia de implementar testes capazes de cobrir 100% do código desenvolvido. Isso acabou acontecendo graças ao fato de que observou-se que tais testes, mesmo cobrindo 100% do código, não chegavam a cobrir 100% das possibilidades que o código cobria, o que fazia com que *bugs* continuassem sendo descobertos mesmo sob o código já testado. Diante disso, todo código mais recente foi projetado de forma a ser testável (evitando acoplação profunda), mas apenas as partes mais importantes foram testadas, tais como os sistemas básicos, apresentados na Seção 4.2.4, que possuem papel fundamental na garantia da estabilidade do sistema.

7.1 Trabalhos Futuros

Com o desenvolvimento do sistema, foi observado a existência de algumas possibilidades de trabalhos futuros que podem ser realizados visando a continuidade deste trabalho:

- Integração direta entre o sistema e sistemas acadêmicos de diferentes universidades através do uso de adaptadores para realização da matrícula através do simulador;
- Desenvolvimento de módulo capaz de computar matrículas válidas de acordo com as regras de uma universidade eliminando (ou ao menos diminuindo) a necessidade de desenvolver um sistema de matrícula próprio;
- Desenvolvimento de uma suite de testes sofisticada para garantir o funcionamento do sistema nas mais diferentes situações;
- Desenvolvimento de um sistema de notificações capaz de trabalhar na escala necessária, sem ter seu desempenho prejudicado.

Referências

ALABBAS, A.; BELL, J. *Indexed Database API 2.0*. [S.l.], 2017. Https://www.w3.org/TR/2017/PR-IndexedDB-2-20171116/. Citado na página 72.

ATER, T. *Building Progressive Web Apps: Bringing the Power of Native to the Browser*. O'Reilly Media, 2017. ISBN 9781491961605. Disponível em: <https://books.google.com.br/books?id=I8o0DwAAQBAJ>. Citado na página 72.

DACIUK, J. et al. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, v. 26, n. 1, p. 3–16, 2000. Disponível em: <https://www.aclweb.org/anthology/J00-1002>. Citado na página 66.

ENDO, B.; RODRIGUES, J.; VERONA, R. *MatrUSP*. Monografia — Instituto de Matemática e Estatística, Universidade de São Paulo, 12 2016. Citado 2 vezes nas páginas 39 e 40.

ERBETTA, G. A. Meuhorário 2: uma aplicação web para simulação de matr'icula. Departamento de Ciência da Computação, 2017. Citado 2 vezes nas páginas 35 e 36.

HICKSON, I. et al. *HTML5*. [S.l.], 2014. Http://www.w3.org/TR/2014/REC-html5-20141028/. Citado na página 71.

IBGE. *Acesso à Internet e à televisão e posse de telefone móvel celular para uso pessoal*. Rio de Janeiro: IBGE, Instituto Brasileiro de Geografia e Estatística, 2015. 90 p. ISBN 9788524044052. Citado na página 45.

JANSEN, R. *Learning TypeScript*. Packt Publishing, 2015. ISBN 9781783985555. Disponível em: <https://books.google.com.br/books?id=odVOCwAAQBAJ>. Citado na página 73.

JR., T. A.; RIVOAL, F.; ETEMAD, E. *CSS Snapshot 2017*. [S.l.], 2017. Https://www.w3.org/TR/2017/NOTE-css-2017-20170131/. Citado na página 73.

LEMIRE, D. et al. Roaring bitmaps: Implementation of an optimized software library. *Software: Practice and Experience*, Wiley, v. 48, n. 4, p. 867–895, Jan 2018. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.2560>. Citado na página 66.

MEW, K. *Learning Material Design*. Packt Publishing, 2015. ISBN 9781785288715. Disponível em: <https://books.google.com.br/books?id=tyDlCwAAQBAJ>. Citado 3 vezes nas páginas 47, 72 e 76.

PIKE, R. The go programming language. *Talk given at Google's Tech Talks*, 2009. Citado na página 59.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. *Unified modeling language reference manual, the*. [S.l.]: Pearson Higher Education, 2004. Citado na página 54.

RUSSELL, A. et al. *Service Workers 1*. [S.l.], 2017. Https://www.w3.org/TR/2017/WD-service-workers-1-20171102/. Citado na página 72.

Apêndices

APÊNDICE A – Artigo Científico no formato SBC

Guru da Matrícula: Um simulador de matrículas multi-institucional e com suporte a análise de pré-requisitos

Fernando J. Mota[1], **Márcio B. Castro**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
– Florianópolis – SC – Brazil

fernando.mota@grad.ufsc.br, marcio.castro@ufsc.br

Abstract. *Enrollment simulators are tools designed to get around certain deficiencies in university enrollment systems that prevent students from finding the best set of classes. However, simulators created so far haven't the ability to work with more than one university program, and few simulators are capable of handling data such as prerequisites and equivalent subjects that help identify whether or not a student can take a particular subject. In this work, it was proposed and implemented a enrollment simulator prototype that meets these requirements, called Guru da Matrícula, and that has low system resource consumption. The results meet the expectations of resource consumption, with low memory usage, and demonstrate the flexibility of using the system when meeting different data sets in different universities.*

Resumo. *Simuladores de matrículas são ferramentas desenvolvidas para contornar certas deficiências nos sistemas de matrículas das universidades que impedem os alunos de encontrar a melhor combinação de turmas. Entretanto, nenhum simulador criado até então possui a capacidade de trabalhar com mais de uma universidade e são poucos os simuladores capazes de lidar com dados como pré-requisitos e disciplinas equivalentes. Esses dados ajudam a identificar se um aluno pode ou não cursar uma determinada disciplina. Neste trabalho, foi proposto e implementado um protótipo de simulador de matrículas que atenda esses requisitos, denominado Guru da Matrícula, e que tenha baixo consumo dos recursos do sistema. Os resultados obtidos atendem as expectativas de consumo de recursos, com baixo uso da memória, e demonstram a flexibilidade de uso do sistema ao atender diferentes conjuntos de dados em diferentes universidades.*

1. Introdução

Muitos dos sistemas disponibilizados hoje pelas universidades são antigos, e não fornecem muitas informações e recursos para escolher a melhor combinação de disciplinas. Em virtude disso, foram desenvolvidos, por iniciativa dos próprios alunos das universidades, ferramentas denominadas simuladores de matrícula, que procuram auxiliar o aluno no processo da rematrícula a partir de dados públicos fornecidos pela própria universidade a respeito das turmas e horários disponíveis no próximo semestre. Para isso, essas ferramentas possuem dois objetivos principais: Reunir diferentes informações públicas fornecidas pelas universidades em um só painel, e também fornecer ferramentas para que o aluno possa encontrar a combinação de turmas que lhe agradem.

Estes simuladores de matrícula, entretanto, não possuem suporte à múltiplas universidades, o que faz com que, para suportar outra universidade, seja necessário copiar e adaptar o projeto afim de adicionar tal suporte, com ajustes tanto na interface quanto em mecanismos para extrair os dados da universidade desejada. Além disso, muitos dos simuladores analisados também não possuem suporte a análises mais avançadas, como, por exemplo, a análise de pré-requisitos, que permite ao estudante saber é possível ou não se matricular em uma disciplina, a partir da lista de disciplinas já cursadas.

Para delimitar o escopo do trabalho, primeiro foi realizado uma pesquisa procurando identificar e analisar alguns dos simuladores disponíveis atualmente (foram analisados os simuladores para 5 universidades distintas), identificando principalmente os principais pontos fracos desses simuladores. Depois, foi realizada uma pesquisa para identificar os recursos mais desejados pelos usuários, feita com 100 estudantes de diferentes universidades. Dessa pesquisa, concluiu-se que os estudantes procuram um simulador de matrículas multi-institucional, com suporte a sugestão contextual de disciplinas por curso e também busca de disciplinas por professores.

Com isso, determinou-se que o objetivo final deste trabalho é desenvolver um sistema que seja capaz de trabalhar com múltiplas universidades, implementando suporte primário aos pré-requisitos das disciplinas - ou seja, realizando desde validação até sugestão de disciplinas para o aluno com base no seu curso e histórico de disciplinas feitas - e também a disciplinas equivalentes. O sistema proposto usará um formato comum, estabelecido anteriormente, para ler os dados das universidades em um banco de dados, de forma genérica. Além disso, serão propostos extratores de dados para duas universidades públicas brasileiras, a Universidade Federal de Santa Catarina, e a Universidade de São Paulo, que serão responsáveis por extrair os dados dos sites das respectivas instituições e armazenar os dados coletados no formato comum previamente especificado.

2. Os Simuladores de Matrícula

Para conhecer mais as opções já disponíveis na internet, foram analisados simuladores de 5 universidades diferentes: UFSC, USP, UFBA, UNICAMP e UTFPR. Em todos os casos analisados, foi possível observar não somente que os simuladores analisados não possuíam suporte a outra universidades, como também o fato de que esse recurso é importante, ao considerar a existência de simuladores copiados e adaptado para suportar outra universidade, como no caso do MatrUSP, que é um *fork* adaptado para atender a USP, desenvolvido a partir do código do MatrUFSC, que originalmente atendia apenas a UFSC.

Para simplificar a análise, apenas o último simulador de cada universidade será analisado aqui, ou seja: MatrUFSC2 (para a UFSC), MatrUSP nova versão (para a USP), Meu Horário 2 (para a UFBA), GDE (para a UNICAMP) e Grade na Hora! (para a UTFPR). Nestes 5 simuladores de matrículas analisados, foram encontrados algumas limitações:

- Apenas 2 (Meu Horário 2 e GDE) dos 5 simuladores analisados possuem suporte à alguma forma de análise de pré-requisitos;
- Apenas 3 (MatrUFSC2, MatrUSP e Grade na Hora!) dos 5 simuladores analisados permitem salvar planos no servidor;

- Apenas 3 (MatrUFSC2, MatrUSP e Meu Horário 2) dos 5 simuladores analisados permitem gerar combinações de turmas;

No geral, foi possível observar que são poucos os simuladores que suportam analisar pré-requisitos das disciplinas afim de verificar se o usuário pode ou não cursar a disciplina. Além disso, a partir dessa análise foi possível levantar uma lista de recursos básicos a serem considerados para implementação no sistema.

3. Estudo Preliminar e Levantamento de Requisitos

Para verificar os recursos que os usuários mais sentem falta, foi realizado uma pesquisa envolvendo 100 estudantes de diferentes universidades questionando, principalmente, quais funcionalidades os alunos mais procuravam em um simulador de matrículas. Desta lista, composta por um total de 13 recursos, foram escolhidos três recursos para serem implementados:

- Busca de disciplinas por professores;
- Sugestão contextual de disciplinas por curso;
- Suporte a outras universidades

Além dos requisitos funcionais, o projeto também possui alguns requisitos não funcionais que o permitirão:

- Consumir menos que 512MB de RAM por servidor (ou seja, no *backend*) durante seu funcionamento;
- Ser compatível com as duas versões mais recentes dos navegadores Google Chrome, Mozilla Firefox, Microsoft Edge e Apple Safari, incluindo também as versões móveis de cada navegador.

4. O protótipo: Guru da Matrícula

O projeto foi desenvolvido em duas partes separadas, mas integradas uma com a outra. De um lado, o *backend* será responsável por tratar os dados recebidos do usuário e das universidades, além de ser responsável pela busca e pelo fornecimento de dados coletados ao usuário. Do outro lado, o *frontend* será responsável por disponibilizar ao usuário uma interface para a visualização desses dados e também para o gerenciamento de planos em qualquer dispositivo (móvel ou não). É possível visualizar essa estrutura na figura 1.

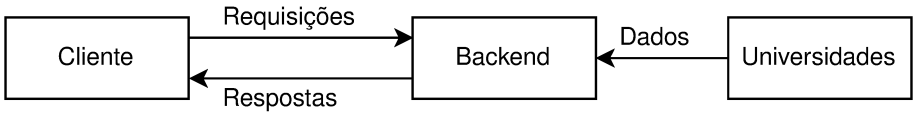

Figura 1. Estrutura geral do sistema

Cada uma dessas partes foi desenvolvida de uma maneira específica. O *backend* foi explicitamente desenvolvido focando em baixo consumo de memória e funcionamento distribuído, de maneira resiliente a falhas. Já o *frontend* foi projetado de forma a transferir a mínima quantidade de dados para o usuário, mas ainda fornecendo recursos como sincronização de dados para uso *offline* e interface adaptada para uso por dispositivos móveis.

4.1. Visão Geral

Por possuir uma série de requisitos complexos, o projeto foi dividido em várias partes, as quais foram integradas corretamente para que o sistema funcione. Numa visão global, é possível perceber uma divisão clara em duas partes: *backend* e *frontend*, desenvolvidas em linguagens diferentes e funcionando em conjunto para formar o resultado final. A Figura 1 mostra como é feita a integração entre o *frontend* e o *backend*.

Nesse modelo, o *frontend* é responsável por servir de interface para o usuário e fazer requisições ao *backend* em busca dos dados com o qual precisa trabalhar. Por exemplo, o *frontend* pode requisitar dados de uma determinada turma, de uma determinada disciplina, de um determinado professor, etc.

Por outro lado, o *backend* é responsável por organizar e otimizar os dados da aplicação, que são fornecidos tanto pelas universidades, na forma de turmas e habilitações para cadastrar, quanto pelos próprios usuários, na forma de planos salvos para um dado semestre. Além de tratar da organização e otimização do armazenamento, é o *backend* que atende as requisições de leitura para os dados armazenados, através do uso de índices de busca, *cache* e versionamento dos dados cadastrados.

4.2. *Backend*

O *backend* foi dividido em duas partes principais: a aplicação, mostrada no bloco (2) da Figura 2, que contem toda a lógica de negócio do sistema, incluindo conhecimento sobre o formato das estruturas armazenadas no banco de dados e sobre o modelo de dados usado na comunicação com o *frontend* (que nada mais é do que um cliente do ponto de vista do *backend*, e portanto é representado no bloco (1)), e também os sistemas básicos, mostrado no bloco (3) que atuam como um *framework*, de forma genérica, fornecendo uma base simplificada e portável para a aplicação. Já os dados de turmas e cursos são fornecidos pelas universidades, representados no bloco (4), que transferem os dados para a aplicação a partir de arquivos *JavaScript Object Notation* (JSON) formatados de acordo com uma estrutura comum, de acordo com o tipo de dado a ser importado.

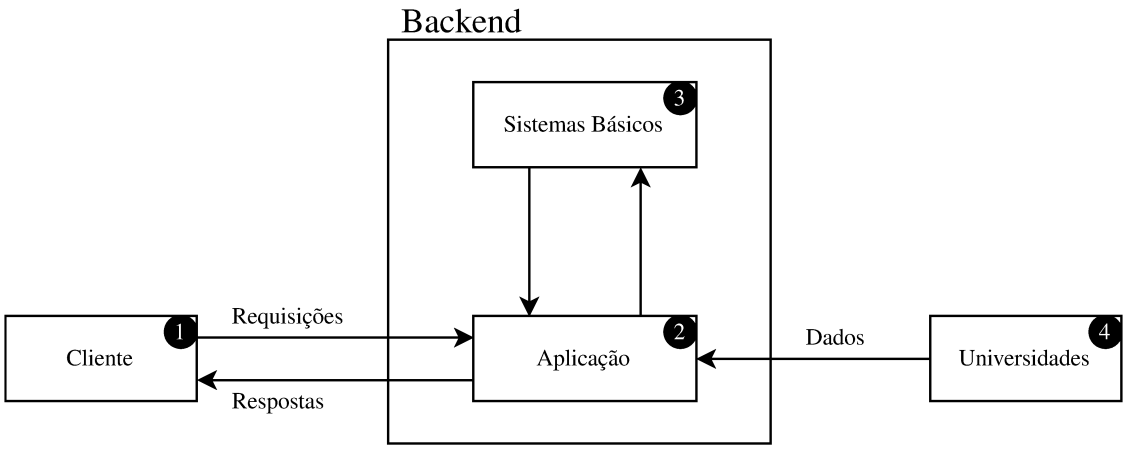

Figura 2. Estrutura geral do *backend*.

O *backend* foi projetado de forma a poder trabalhar com diferentes plataformas ao mesmo tempo que fornece um conjunto uniforme de recursos. De modo geral, o *backend*

é responsável, entre outras coisas, por:

- Realizar o cadastro dos dados de turmas e horários a partir dos dados fornecidos pelas universidades (ou por *web scrapers*) periodicamente;
- Realizar autenticação (*login*/*logout*) do usuário;
- Armazenar os planos salvos pelo usuário associando com sua respectiva conta;
- Armazenar dados relacionados ao usuário tal como universidade em que estuda, curso que faz e disciplinas que já estudou;
- Realizar o versionamento dos dados disponibilizados e disponibilizar as mudanças em cada "versão" para cada curso, de cada universidade, de forma separada;
- Fornecer uma *Application Programming-Interface* (API) *GraphQL* para disponibilização dos dados da base de dados e também salvamento de dados de planos salvos por usuários;
- Realizar indexação e permitir busca dos dados armazenados através de diferentes filtros, tais como nome da disciplina ou professor que ministra as aulas.

Observe que, a princípio, o *backend* não é responsável por capturar e processar diretamente os dados das universidades (isto é, ele não busca interpretar o conteúdo das páginas de um sistema acadêmico). Em vez disso, o *backend* apenas executa requisições para programas separados, os quais podem ser desenvolvidos em linguagens diferentes, que então fornecem os dados em um formato adequado para o processamento do sistema, baseado no formato JSON mas adaptado para as estruturas de dados necessárias. Isso foi desenvolvido de forma que novas universidades possam ser adicionadas de forma simples, sem ser necessário conhecimento avançado do próprio funcionamento interno do *backend*, o que permite, inclusive, que a própria universidade forneça diretamente os dados, bastando apenas implementar o formato de dados JSON.

Nos casos das universidades que serão abordadas nesse trabalho (Universidade Federal de Santa Catarina (UFSC) e Universidade de São Paulo (USP)), os dados serão capturados através de *web scrapers* desenvolvidos na mesma linguagem do *backend* que, a partir dos dados disponibilizados pelos sistemas acadêmicos, realizarão a conversão para o formato de dados JSON esperado pelo sistema. Tais mecanismos serão abordados na Seção 5.

4.2.1. Linguagens de Programação

O *backend* foi desenvolvido em **Go**[Pike 2009], uma linguagem de programação de código aberto desenvolvida pelo Google que tem como características o fato de ser concorrente, possuir compilação rápida e ter coleta de lixo (*garbage collection*). Esta linguagem possui grande semelhança com a linguagem **C** em termos de sintaxe, mas com a adição de comandos para programação concorrente e a remoção de parênteses em torno de estruturas como *for* e *if*.

Essa linguagem foi escolhida por possuir como características a capacidade de gerar binários estáticos como resultado da compilação. Além disso, possui baixo consumo de memória e é compatível com uma ampla variedade de arquiteturas, como ARM, x86, MIPS e afins, e também sistemas operacionais, como Linux, Mac OS X e Windows. A linguagem Go também possui uma grande biblioteca padrão e um grande ecossistema

de ferramentas que fornece recursos como detecção de tratamento de erros, formatação do código (algo que a linguagem obriga nativamente ao compilar o código) e também detecção de possíveis *bugs*, facilitando assim o desenvolvimento de projetos de variados tamanhos.

Por fim, ela é uma linguagem capaz de aproveitar bem os processadores ou núcleos de processamento do servidor onde o programa é executado, pois ela permite dividir o trabalho entre inúmeras *goroutines* que são agendadas eficientemente pelo próprio ambiente de execução do Go onde o programa executa. Isso faz com que a linguagem seja perfeita para o desenvolvimento de programas que criam servidores *web* (como é o caso deste projeto), pois permite que cada requisição recebida seja tratada eficientemente e também que milhares de requisições possam ser processadas ao mesmo tempo. Por estas razões, a linguagem Go se torna mais eficiente do que outras linguagens interpretadas, tais como Python, PHP e Ruby.

4.2.2. Plataformas Suportadas

Devido à natureza multi-plataforma da linguagem de programação escolhida, é possível executar o projeto em todas as plataformas que o compilador suporta nativamente, bastando apenas fazer o *cross-compiling*[1] para essas outras plataformas.

Em relação às dependências da própria plataforma, como banco de dados e sistema de *cache*, o *backend* continua sendo compatível com as plataformas que o próprio Go suporta, mas também contando com suporte para outras plataformas, como é o caso do ecossistema do Google App Engine (GAE), o qual bloqueia escritas no próprio sistema de arquivos da máquina, incentivando assim o uso de serviços apropriados para as tarefas, como o *Google Cloud Storage*[2] e o *Google Cloud Datastore*[3].

Isso significa que o sistema será capaz tanto de executar em um microcomputador, quanto em um *Raspberry Pi* ou mesmo em um ambiente distribuído como o GAE, usando as APIs apropriadas através do uso de abstrações no *software*. Ou seja, através da implementação de um pouco de código, será possível fazer com que o sistema suporte outro Sistema Gerenciador de Banco de Dados (SGBD) facilmente.

Um aspecto importante a respeito dessas abstrações é que elas não tem um uso global sobre todo o sistema. Ou seja, elas são injetadas nos componentes que precisam delas durante a inicialização desses componentes, o que permite maior flexibilidade e eficiência ao fazer que componentes diferentes possam usar dependências diferentes conforme suas necessidades.

4.2.3. Sistemas básicos

A aplicação depende de alguns sistemas básicos, cuja estrutura está destacada na Figura 3, que fornecem a base para o funcionamento do sistema, como uma espécie de *framework*. Entre os serviços fornecidos, estão acesso a banco de dados (bloco (7)), *cache* (bloco

[1] Compilação para uma plataforma que não é a mesma onde o compilador está executando.
[2] *Google Cloud Storage* fornece armazenamento e leitura de arquivos com redundância e confiabilidade.
[3] *Google Cloud Datastore* é um banco de dados NoSQL altamente distribuído e escalável.

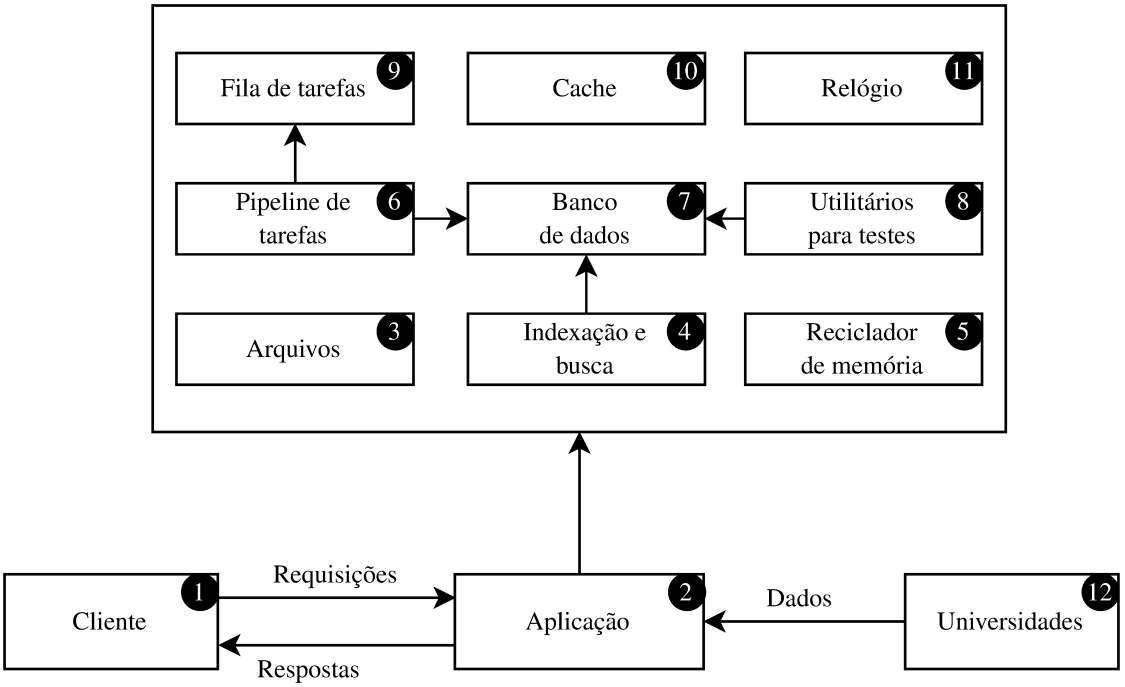

Figura 3. Estrutura geral dos sistemas básicos.

(10)), gerenciamento de arquivos (bloco (3)), indexação de dados e busca (bloco (4)), fila de tarefas (bloco (9)), *pipeline* de tarefas (bloco (6)), entre outros, menos importantes mas ainda úteis, como o relógio (bloco (11)), um reciclador de memória (bloco (5)) e alguns utilitários para testes (bloco (8)), que atuam de forma genérica, sem conhecimento interno das características do resto da aplicação (representado no bloco (2)), e que podem ser usados de forma também genérica, com o uso de adaptadores permitindo a substituição dos componentes sem a necessidade de refatorar todo o código desde que implementado as interfaces desses sistemas básicos.

Graças a esses sistemas básicos é que é possível obter, além de portabilidade, maior performance, ao facilitar a implementação de algoritmos mais eficientes. Um exemplo é a importação de dados das universidades (representadas no bloco (12)), que é uma tarefa realmente intensiva em termos de quantidade de dados, e que é facilmente paralelizada com o uso da *pipeline* de tarefas (representada no bloco (6)), que é utilizado pela aplicação (bloco (2)) para diminuir o tempo necessário para importar os dados. Além disso, outro exemplo é o atendimento à requisições do cliente (bloco (1)), que se aproveitam bastante tanto da implementação de *cache* (bloco (10)), quanto do sistema de indexação e busca (bloco (4)).

Essas características permitem que esses sistemas básicos possam ser usados em outros projetos e obter essas mesmas características de portabilidade e foco em performance que este sistema possui.

4.2.4. Aplicação

A aplicação é dividida em 3 partes principais: uma para realizar o controle de operações relacionadas a usuários (bloco (4) da Figura 4) - como cadastro e autenticação - outro para realizar a importação dos dados (bloco (5)) das universidades (bloco (6)), e, por fim, outro para permitir o acesso a esses dados pelo cliente, através de uma API pública *GraphQL* (bloco (3)).

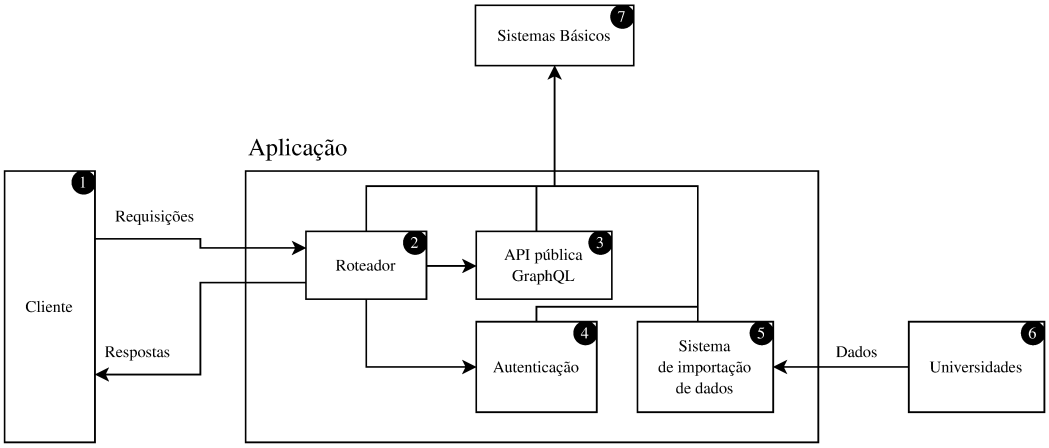

Figura 4. Estrutura geral da Aplicação.

Além disso, a aplicação também possui uma parte auxiliar, que é responsável por atender as requisições do cliente (bloco (1) da Figura 4) e direcioná-las para o sub-sistema correto, e é denominada roteador (bloco (2)),

Este sistemas são dependentes de abstrações extras que usam os sistemas básicos (bloco (7) da Figura 4) apresentados na Seção 4.2.3 para, entre outras coisas, realizar o processamento e a persistência dos dados. São essas abstrações que garantem a consistência e a padronização dos dados salvos, permitindo que o dado escrito possa ser lido em um mesmo formato otimizado para armazenamento em banco de dados, por exemplo.

O sistema de autenticação, por exemplo é, basicamente, uma série de estruturas integrando a biblioteca *Authboss*[4] (bloco (2) da Figura 5) com o resto da aplicação (bloco (3)) e fornecendo persistência a partir do sistema básico de banco de dados (bloco (4)), de forma a fornecer, entre outros recursos, cadastro, *login* e *logout* para o cliente (bloco (1)).

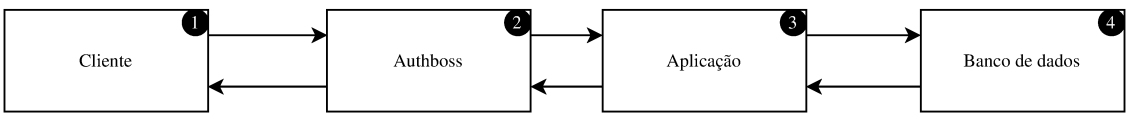

Figura 5. Estrutura do sistema de autenticação

Através desta biblioteca, o sistema de autenticação provê suporte a cadastro, autenticação e *logout*, tanto a partir do tradicional conjunto de *e-mail* e senha quanto

[4] https://github.com/volatiletech/authboss

a partir da integração, via OAuth2, com sites como Google e Facebook. Dados importantes, como senhas, são codificados usando **BCrypt**, o que garante a segurança desses dados em caso de invasão ou vazamento dos dados do banco de dados, por exemplo.

Outro sistema importante é o sistema de importação de dados, que é um mecanismo distribuído, composto por componentes conectados entre si que realizam, de maneira otimizada, a inserção, atualização e remoção de dados do banco de dados (representado pelo bloco (8) da Figura 6) a partir dos arquivos JSON criados seguindo um padrão comum (que varia de acordo com o tipo de dado a ser importado, que pode ser tanto dados de turma, quanto dados de habilitações) a partir dos dados disponibilizados pelas universidades (bloco (1)). Ou seja, esse sistema, por si só, **não lida** com a extração dos dados das universidades. Em vez disso, é esperado um arquivo JSON padronizado, cujo *download* é realizado na etapa (3) do processo de importação.

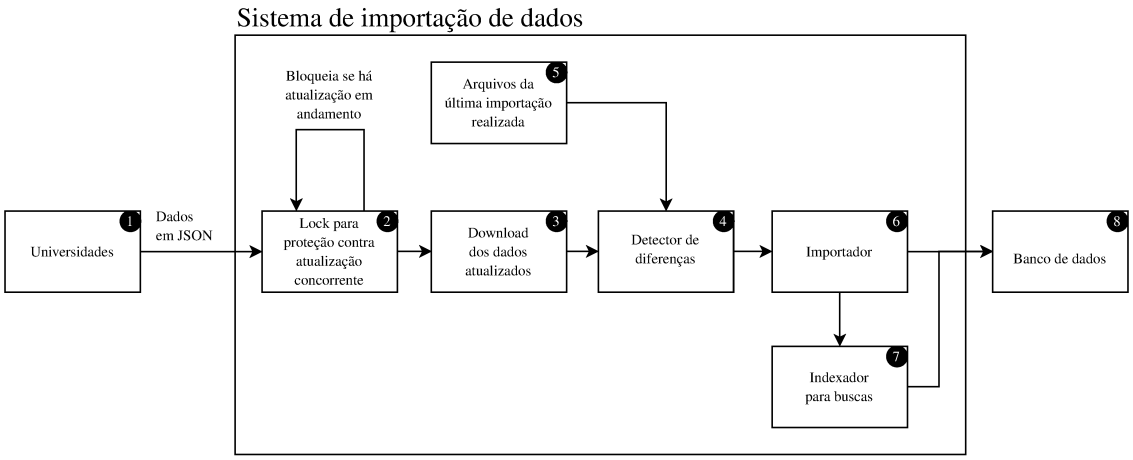

Figura 6. Estrutura geral do sistema de importação de dados.

Embora seja distribuído, esse sistema não suporta atualizações concorrentes para um mesmo tipo de dado, graças à necessidade de lidar com arquivos da última importação realizada com sucesso (demonstrado no bloco (5) da Figura 6) para poder detectar, no bloco (4), as diferenças entre cada arquivo, ou seja, o que foi de fato inserido, atualizado ou removido entre os arquivos, afim de poder fazer a atualização dos dados no banco de dados e também no índice de busca, nas etapas (6) e (7).

Devido a essa limitação, há uma etapa, representada no bloco (2) da Figura 6, onde é realizada uma checagem para garantir que não há nenhuma tarefa de importação para uma dada universidade e tipo de dado (no caso, se é oferta de turmas ou se são dados de cursos) executando no momento, através de um *Lock* distribuído usando os aspectos transacionais do sistema básico de banco de dados. Só quando a tarefa até então em execução terminar é que é disparado um novo processo de atualização, o que garante tanto a consistência do processo de importação quanto alivia a carga sob o sistema.

As etapas (6) e (7), mostradas na Figura 6, são realizadas de maneira paralelas de acordo com a própria natureza das atualizações. Tabelas são atualizadas de maneira totalmente paralela, por exemplo, dado que cada registro emitido pelo detector de diferenças se refere a exatamente um registro da tabela, que pode ser inserido, atualizado ou remo-

vido. Já os índices são, cada um, atualizados de maneira sequencial, com as remoções acontecendo primeiro, registros novos sendo inseridos em seguida e os registros modificados sendo atualizados por último, mas índices diferentes são processados de maneira paralela conforme possível. Todos esse processamento, no final, acaba resultando em operações para o sistema básico de banco de dados.

Ao final do processo de importação, o registro da própria universidade processada é atualizado com dados como última atualização e lista de arquivos da importação recém-atualizada. Além disso, o "dono" da proteção contra atualização concorrente é apagado, o que permite que a próxima tarefa de importação seja executada logo na sequência.

Por fim, o sistema possui uma API pública *GraphQL*, que é a API principal do sistema, e que é responsável por permitir o acesso a todos os dados coletados pelo sistema de importação. A única parte não totalmente exposta sob *GraphQL* é a API de autenticação, que depende de *endpoints* próprios para permitir autenticação *OAuth2* (ou seja, com Facebook e Google), entre outras ações.

A API *GraphQL* é exposta a partir de um modelo pré-definido, que define todos os tipos, campos, métodos e seus argumentos que podem ser usados nas consultas em *GraphQL*. Para criar a API, foi utilizado o projeto **gqlgen**, representado no bloco (2) da Figura 7, que gera código Go baseado no modelo *GraphQL*, e que auxilia na "tradução" das requisições GraphQL em chamadas à aplicação, que então consulta o banco de dados (bloco (5)) e retorna os dados, que são codificados de acordo com o padrão *GraphQL*, para o cliente (bloco (1)).

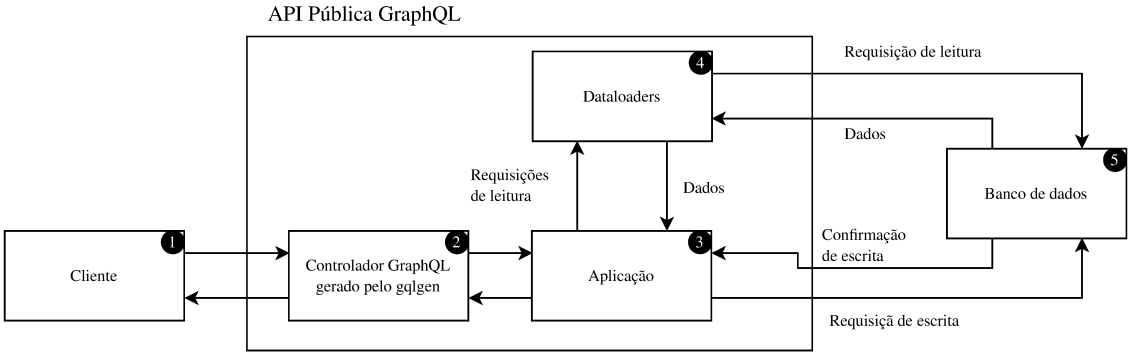

Figura 7. Fluxo de atendimento de consultas *GraphQL*.

Quando a requisição *GraphQL* é somente-leitura, a aplicação, em vez de consultar diretamente o banco de dados, faz as requisições usando um *dataloader* (bloco (4) da Figura 7), que, entre outras propriedades, provê um *cache* temporário (apenas durante a duração da requisição) e permite agrupar diferentes consultas paralelas ao banco de dados (na mesma requisição) em uma só consulta, o que permite reduzir a latência de comunicação com o banco de dados em SGBD que suportem a leitura de múltiplos registros em uma só consulta.

Quando a requisição *GraphQL* é uma mutação, ou seja, altera algum registro do banco de dados, tal otimização não é possível, e então a aplicação faz a requisição de atualização diretamente ao banco de dados, como mostra a Figura 7, com a conexão direta entre os blocos (3) e (5).

Em ambos os casos, fica sob responsabilidade do controlador gerado pelo *gqlgen* a validação dos tipos da requisição - de acordo com o definido no modelo *GraphQL* - e fica sob a responsabilidade da aplicação a validação final da requisição, através da verificação da autorização do usuário para fazer a requisição, por exemplo, de forma a impedir que usuários possam modificar ou mesmo ler registros sob os quais não tem as respectivas permissões.

Em termos de performance, o fato do *gqlgen* ser um gerador de código permite que o compilador da linguagem otimize o resultado final, o que compensa parcialmente o custo do *parsing* das requisições *GraphQL*. Além disso, se um usuário adiciona mais de uma consulta à uma única requisição *GraphQL*, tais consultas são executadas paralelamente, o que ajuda a diminuir o tempo de resposta consideravelmente.

4.3. *Frontend*

O *frontend* do projeto foi desenvolvido de forma a ser leve (menor do que 100KB no carregamento inicial) e responsivo entre vários dispositivos. Essa parte do sistema é a parte visível ao usuário e funciona apenas em navegadores recentes, usando recursos baseados em HTML5[Hickson et al. 2014]. Dentre as características do *frontend*, estão:

- Visual baseado em *Material Design*[Mew 2015];
- Suporte a *download* de código sob demanda;
- Salvamento de dados para acesso *offline* usando *Service Workers*[Russell et al. 2017] e *IndexedDB*[Alabbas and Bell 2017], com suporte a sincronização de mudanças incrementais;
- Geração de combinações de turmas sob demanda, com suporte à filtragem das combinações geradas;
- Emissão de avisos tais como falta no cumprimento de pré-requisitos e conflitos de horários;
- Interface para administração de contas do usuário, incluindo opções para configurar o curso que o usuário faz, o campus e universidade em que estuda, pré-requisitos já cumpridos, e configurações relacionadas a salvamento de dados *offline*.

O *frontend* foi desenvolvido e testado primariamente em celulares e então adaptado para funcionamento com dispositivos que possuem telas maiores, tendo sido testado exaustivamente para permitir sua correta execução a 60 Quadros por Segundo (QPS), mesmo em celulares, mantendo a fluidez e responsividade no uso. Por fim, o *frontend* é um *Progressive Web App* (PWA) [Ater 2017], ou seja, uma aplicação capaz de se adaptar progressivamente aos recursos disponíveis no navegador do usuário. Na prática, isso quer dizer que a aplicação será:

- **Confiável**: carregamento instantâneo, independente das conexões de rede, após o primeiro acesso;
- **Rápida**: responde rapidamente as interações de usuário e sem travamento mesmo em ações simples como *scroll* de página;
- **Atraente**: similar a um aplicativo no dispositivo com experiência de usuário imersiva.

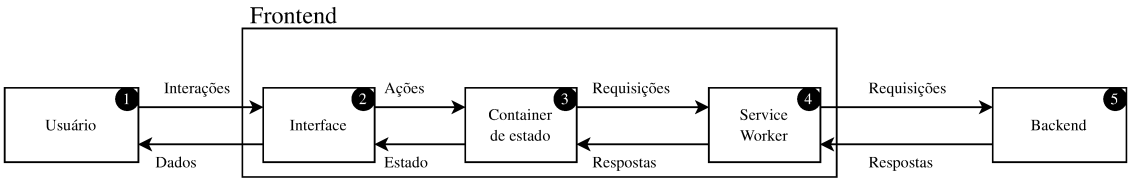

Figura 8. Estrutura geral do *frontend*.

Para que tudo isso fosse possível, o *frontend* foi dividido em três partes principais, conforme mostrado na Figura 8: a **interface**, mostrada no bloco (2), com a qual o usuário (bloco (1)) interage e visualiza diretamente, o ***container* de estado**, mostrado no bloco (3), que contem a lógica das ações associada com a interface, e o ***Service Worker***, mostrado no bloco (4), que intercepta as interações entre a página (composta por interface e *container* de estado) e o *backend* (mostrado no bloco (5)) e permite que toda a aplicação funcione mesmo sem conexão com a internet, através do armazenamento de dados em um banco de dados local no navegador do usuário.

4.3.1. Linguagens de Programação

O *frontend* foi desenvolvido em **Typescript**[Jansen 2015], uma linguagem desenvolvida pela Microsoft que é um superconjunto com suporte a tipos do Javascript, linguagem muito usada para adicionar interatividade em páginas *web*. Como o Typescript compila para Javascript, o resultado final para o usuário será o mesmo e qualquer navegador com os recursos necessários e Javascript habilitado será capaz de executar o código.

A escolha pelo Typescript surge do fato de que o Javascript por si só é uma linguagem dinâmica e sem tipos. O Typescript adiciona suporte avançado a tipagem na linguagem, permitindo que muitas inconsistências sejam detectadas imediatamente durante a compilação do código e não apenas em tempo de execução (*runtime*). Além disso, como o resultado efetivo gerado é apenas o código Javascript sem os tipos anexados, o resultado final é o mesmo, não havendo custo adicional de processamento para o usuário devido ao uso da linguagem.

Em termos de estilo, o projeto usa *Sassy CSS* (SCSS), que é uma linguagem que compila para *Cascading Style Sheets* (CSS) [Jr. et al. 2017], esta, sim, suportada por todos os navegadores modernos e que permite definir características básicas de estilo de texto, tais como posicionamento, cor de fundo e cor do texto. A vantagem do uso do SCSS para definição dos estilos é a integração com as bibliotecas de estilo e a maior facilidade de customização através do uso de variáveis e funções auxiliares (que são corretamente transformadas para CSS puro durante a compilação). Em relação às bibliotecas de estilo, o *frontend* usa uma biblioteca chamada *RMWC* em conjunto com a *material-components-web*, que implementa a linguagem de *design* chamada *Material Design*, o que permite ao sistema ter um visual respeitando essa linguagem de *design*.

4.3.2. Plataformas Suportadas

Em relação as plataformas suportadas, o *frontend* possui um conjunto limitado de navegadores suportados. Não serão suportados navegadores antigos e/ou abandonados, tais como Internet Explorer. Em vez disso, serão suportados apenas as duas últimas versões estáveis de cada navegador, mas sem garantir acesso a recursos avançados como acesso *offline*. Felizmente, como os navegadores hoje contam com suporte a atualização automática, o número de pessoas usando navegadores atualizadas é bem grande. Segundo estatísticas globais publicadas pelo site *StatsCounter*[5], esse limite compreende mais de 60% do *marketshare* global de navegadores.

Além disso, graças ao fato de que esses navegadores suportam os recursos mais recentes em termos de Javascript e CSS, é possível diminuir muito o tamanho do código final entregue ao usuário ao evitar o uso de *polyfills* e *shims* criados para suportar navegadores antigos.

4.3.3. Interface

A interface foi desenvolvida usando *React*[6], que é uma biblioteca Javascript extremamente otimizada para desempenho baseada na ideia de *virtual-DOM*, onde o sistema retorna a estrutura desejada para o componente de acordo com um estado específico, e a própria biblioteca se encarrega de detectar as diferenças em relação ao que já está sendo exibido e, então, realizar apenas o conjunto mínimo de operações necessárias para que o componente exibida corresponda exatamente ao que a aplicação retornou.

Essa abordagem proporciona benefícios tanto em desempenho quanto em confiabilidade ao evitar que a aplicação precise manipular, diretamente, os elementos visuais correspondentes ao que está sendo atualizado, o que acaba reduzindo a complexidade da aplicação e portanto seu tamanho. Desta forma, é projetar componentes que são independentes do resto do aplicativo, e também mais fáceis de testar, uma vez que a estrutura retornada pelo componente depende apenas da entrada de dados e do estado interno (que tende a ser pequeno, por se tratar de apenas uma pequena parte da página), do próprio componente, que pode ser facilmente reproduzido em qualquer ambiente *Javascript*.

Em relação à própria estrutura interna da interface, os componentes são implementados de maneira a serem reusáveis sempre que possível, possibilitando reaproveitar, por exemplo, componentes que aparecem em mais de uma página do sistema, como calendários e menus laterais. Isso ajuda a diminuir a quantidade de código total necessária para renderizar a interface, além de simplificar o código e implementação.

Tais componentes são carregados de forma dinâmica, sob-demanda, dependendo da página acessada pelo usuário, o que ajuda a diminuir ainda mais o tamanho do código enviado para execução pelo navegador do usuário. Essas páginas, representadas no bloco (2) da Figura 9, são estruturadas como componentes especiais que, diferente dos demais componentes reusáveis (bloco (3)), são projetados para lidar com aspectos próprios do conteúdo a ser exibido, como consultas ao servidor (ou *service worker*, como mostrado

[5] http://gs.statcounter.com/browser-version-market-share
[6] https://reactjs.org/

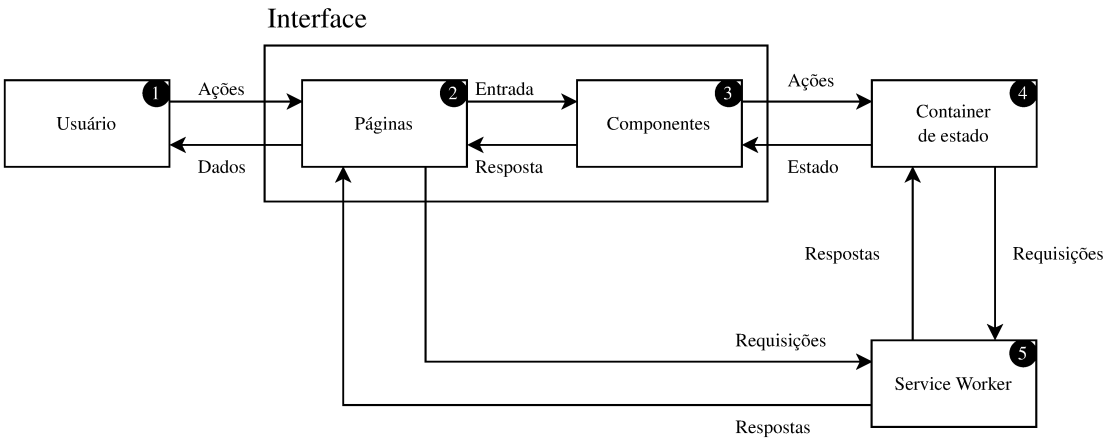

Figura 9. Estrutura geral da *interface*.

no bloco (5)) e também leitura de parâmetros da URL, o que faz com que tais páginas operem de maneira mais específica e menos dependente do componente. Para ajudar na manutenção do estado entre os componentes, é utilizado um *container* de estado (bloco (4)) que permite sincronizar o estado compartilhado entre diferentes componentes, que também é capaz de realizar requisições diretamente para o *backend/service worker* (bloco (5)).

Por fim, todo o visual da interface foi baseado em *Material Design*[Mew 2015], implementado no código a partir de uma biblioteca chamada *RMWC*[7], que implementa componentes reusáveis respeitando as regras visuais do *Material Design* para *React*, através do uso da biblioteca oficial para *web*, chamada *material-components-web*[8].

As páginas de autenticação, por exemplo, são, em sua maioria, páginas públicas, que fornecem ao usuário a opção de fazer cadastro, *login* e recuperação de senha. Por sua funcionalidade, essas páginas tem como característica estar centradas, principalmente, em formulários, que requisitam do usuário as informações que são necessárias para que a funcionalidade seja implementada. Junto a esses formulários, são mostrados também textos auxiliares escritos para guiar o usuário durante o processo, com informações que possam ajudá-lo a entender o foco do formulário, além de disponibilizar links para outras páginas úteis (por exemplo, ao permitir que o usuário acessando a página de *login* possa acessar a página de cadastro facilmente, e vice-versa).

A página de cadastro, mostrada na Figura 10, é uma página com um formulário contendo quatro campos: nome, *e-mail*, senha e confirmação de senha. A partir dessa página, é permitido ao usuário se cadastrar o conjunto de *e-mail* e senha, evitando portanto que o usuário precise ter contas em outros serviços e ainda fornecendo a possibilidade, ao usuário, de usar uma senha diferente para uso do serviço. Nessa página não é fornecido opções para cadastro usando serviços de terceiros pois, ao fazer o *login* pela primeira vez, caso ainda não cadastrado, o usuário usa a API do serviço a partir do qual o usuário se conectou para obter as informações necessárias para completar o cadastro, como nome e *e-mail*, por exemplo. Entretanto, são fornecidos links de acesso rápido às páginas de *login*

[7] https://rmwc.io
[8] https://github.com/material-components/material-components-web

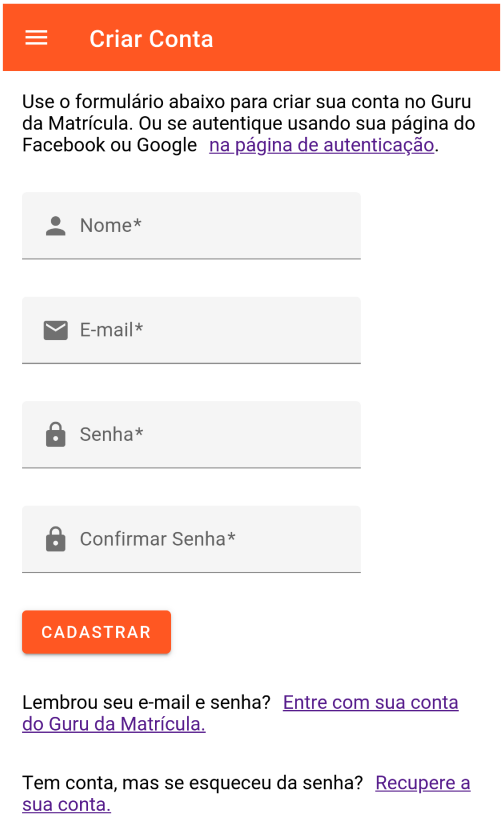

Figura 10. Página de cadastro.

e de recuperação de senha, para caso o usuário observe que não precisa, de fato, criar uma nova conta para acessar o sistema.

Já a página de *login*, mostrada na Figura 11, é uma das páginas mais complexas entre as páginas de autenticação. Nela, é possível fazer *login* a partir de qualquer uma das variadas opções de autenticação suportadas pelo sistema, que incluí o tradicional conjunto de *e-mail*/senha, assim como *login* através de serviços de terceiros, como Google e Facebook.

No caso desses serviços de terceiros, como é necessário abrir páginas externos ao site, tais páginas são abertas usando *pop-ups* que, então, permitem que o usuário entre com os dados para o site externo e, ao final do processo, retorne automaticamente ao site, que detecta automaticamente problemas no processo. Por meio dessa opção de *login* senhas não são trocadas, o que garante a segurança do processo. Já no caso de *login* usando usuário e senha, ao enviar o formulário o sistema faz uma requisição *Asynchronous Javascript And XML* (AJAX) para o servidor *Hypertext Transfer Protocol* (HTTP) (especificamente para o endereço no qual o *Authboss* atende) com as credenciais e, em caso de erro, avisa o usuário. Em caso de *login* bem sucedido, o usuário é redirecionado para a página definida como parâmetro do endereço da página ou, caso nenhum parâmetro tenha sido especificado, para a página inicial do site. Em caso de falha uma mensagem é mostrada para permitir que o usuário tome providências e eventualmente repita o processo.

Figura 11. Página de *login*.

Por fim, a página de recuperação de senha, mostrada na Figura 12, é a página mais simples entre as páginas de autenticação, possuindo apenas um formulário com apenas um campo, para que o usuário entre com o *e-mail* usado no cadastro da conta, de forma que o sistema possa enviar um *e-mail* para esse usuário fornecendo um *link* especial que, quando clicado, permite que o usuário mude a senha usada na conta. Além desse formulário, também são fornecidos *links* de acesso rápido para as páginas de *login* e de *cadastro*.

Já as páginas de planos são, facilmente, as páginas mais acessadas pelos usuários, pois tem como função permitir a administração de planos contendo todo o planejamento para um dado período letivo e universidade.

As duas páginas mais proeminentes dentre as páginas relacionadas a planos são: a página de calendário, mostrada na Figura 13, onde é possível visualizar todo o conjunto de horários e também as turmas escolhidas para uma determinada combinação, e a página de busca, mostrada na Figura 14, que permite adicionar novos registros (sejam disciplinas ou ofertas de disciplinas) ao plano. Ambas as páginas tem diferentes variações, de forma a permitir que o usuário possa explorar tanto diferentes dados (ao buscar por turmas ou buscar diretamente por disciplinas, por exemplo), como também visualizar tais dados de diferentes modos (ao visualizar apenas 1 ou 3 dias da semana, em vez de 7, no calendário, por exemplo).

Também há páginas para visualizar e gerenciar as disciplinas e ofertas de disciplinas adicionadas ao plano, junto a outras páginas com funcionalidades variadas, como permitir gerenciar configurações, como nome do plano, *status* de publicação e outras opções de configuração, ou ainda visualizar versões antigas do plano.

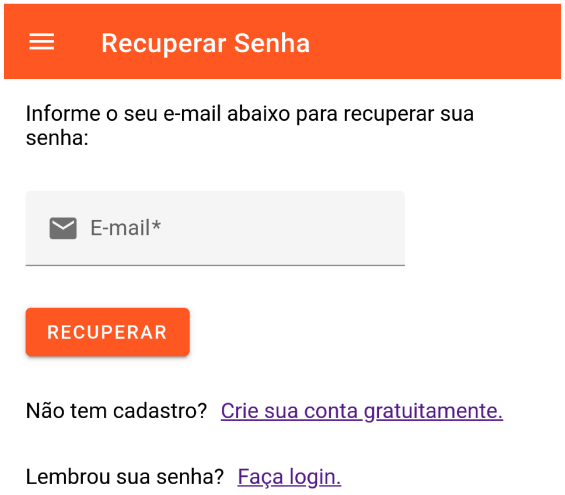

Figura 12. Página de recuperação de senha.

Já as páginas de universidades são umas das menos acessadas pelos usuários, uma vez que tem como utilidade permitir o gerenciamento das universidades cadastradas no sistema, algo que não deverá ser de grande interesse por parte dos usuários visto que o foco, e também a maior parte dos usuários, do sistema são estudantes. Esse procedimento inclui, entre outras coisas, ter conhecimento de duas tarefas básicas: como extrair os dados de turmas e cursos da universidade, e também como formatar todos esses dados no formato JSON, na estrutura exigida pelo sistema de importação de dados.

A primeira página desse conjunto é a lista de universidades, mostrada na Figura 15, que tem como função listar todas as universidades que o usuário possuí, além de fornecer acesso rápido para as páginas de edição de universidade e também de cadastro de universidades. Nessa página também são mostrados dados associados a cada universidade, como data e hora da última atualização correspondente a cada universidade. Note que, após cadastrada, uma universidade não pode ser apagada, mas pode ser ocultada para os demais usuários.

Outra página importante é a página de cadastro de universidade, mostrada na Figura 16, que permite o cadastro de uma universidade no sistema. Para isso, são solicitados informações básicas, como nome e acrônimo, além de informações mais detalhadas tanto para dados de turmas quanto para dados de cursos, como endereço de download do arquivo em formato JSON (formatados conforme o formato esperado), endereço para saber mais sobre os dados baixados e, também, endereço de notificação para apagar o arquivo dos dados originalmente baixados. Ao enviar o formulário, o sistema cadastra a universidade e retorna dois códigos secretos: um para dados de turmas, e outro para dados de curso. Esses códigos secretos devem ser usados para poder comunicar o sistema de importação de dados sobre a disponibilidade dos arquivos de dados para download, e não devem ser compartilhados, pois poderiam permitir a entrada de dados errados no sistema. A universidade cadastrada é salva com visibilidade oculta (não-pública), de forma que é possível entrar com os dados no sistema antes de finalmente disponibilizar a universidade para uso pelos demais usuários.

Figura 13. Página de calendário mostrando combinação encontrada de plano.

Por fim, a página de edição de universidade, é muito similar à página de cadastro de universidade, mas com apenas a adição de três campos: um para permitir que a universidade seja liberada para acesso pelas outras universidades, e outros dois para permitir que novos códigos secretos sejam gerados tanto para o envio de dados de turmas quanto para o envio de dados de cursos. Após o envio do formulário, são mostrados os respectivos códigos secretos solicitados, caso o campo em questão tenha sido marcado, o que garante que o usuário tenha a flexibilidade de mudar a chave de segurança sempre que desejar.

Finalmente, a página de perfil, mostrada na Figura 17, é uma das páginas mais complexas e importantes do sistema. Sua função é permitir que o usuário possa definir dados relacionados ao seu próprio cadastro, como nome, *e-mail*, cursos, habilitações e também as disciplinas que já fez. Além disso, há a opção de configurar a senha, para usuários cuja conta foi criada usando o conjunto de *e-mail* e senha, uma vez que contas criadas a partir de serviços externos não podem ter senha devido a sua própria natureza de depender da autenticação pelo serviço externo em questão.

A página mais complexa relacionada às páginas de perfil é o de seleção de disciplinas já feitas, mostrada na Figura 18 que precisa permitir ao usuário lidar com uma grande quantidade de dados, e por isso faz consultas ao servidor visando economizar na quantidade de dados transferidos ao mesmo tempo que facilita a administração das opções por parte do usuário.

4.3.4. Container de estado

O *container* de estado é constituído por um conjunto de partes que são carregados sob demanda de acordo com a página em que o usuário está e que, utilizando a biblioteca

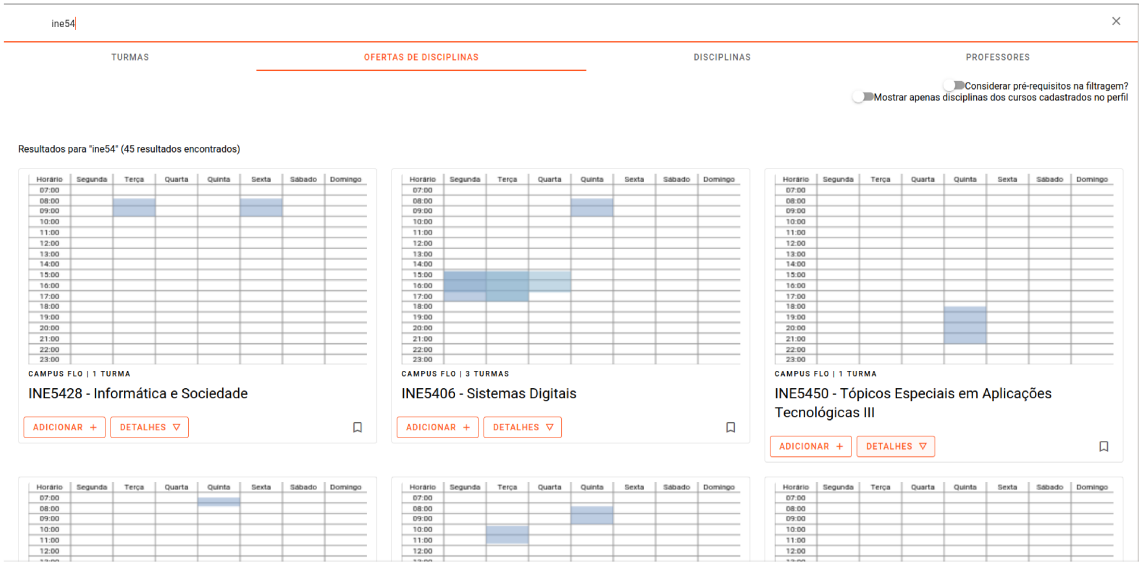

Figura 14. Página de busca de ofertas de disciplinas.

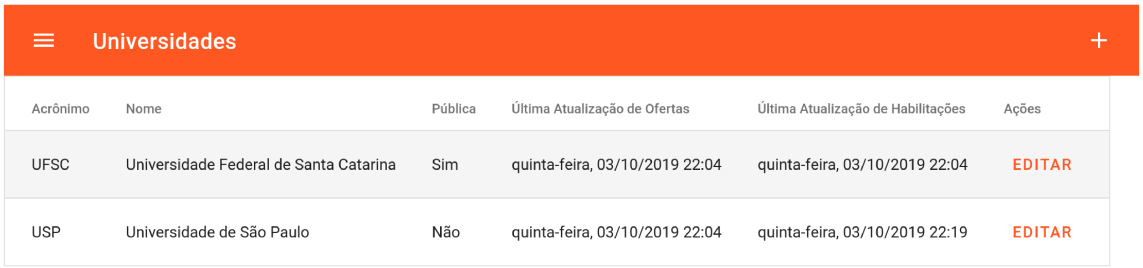

Figura 15. Lista de universidades.

Redux[9], implementam um repositório responsável por armazenar o estado do *frontend*. Por usar *Redux*, tem-se que o estado é imutável, com novos estados sendo gerados de acordo com ações normalmente disparados por ação do usuário, computados por funções denominadas *reducers* (bloco (2) da Figura 19) que, dado o estado anterior e uma ação (que pode ser proveniente do usuário ou de qualquer outro sub-sistema), computam e retornam um novo estado.

Devido a isso, a lógica da aplicação fica protegida contra atualizações concorrentes e é, desta forma, consistente, facilitando tanto a implementação de testes quanto a eventual reprodução de *bugs* e outras falhas na lógica. Além disso, ganha-se acesso a ferramentas que possibilitam, entre outras coisas, a análise das ações registradas até então e também do estado do repositório após efetuar cada ação, o que facilita muito o desenvolvimento. Por fim, ao estruturar-se dessa forma, a lógica do *frontend* acaba por ficar isolada da camada de interface (representada no bloco (1) da Figura 19), o que significa que é possível mudar toda a interface - incluindo bibliotecas e estruturas - sem a princípio ser necessário alterar a lógica de estado do *container* de estado.

Junto ao *container* de estado são executadas funções *Sagas*, representados no

[9] https://redux.js.org/

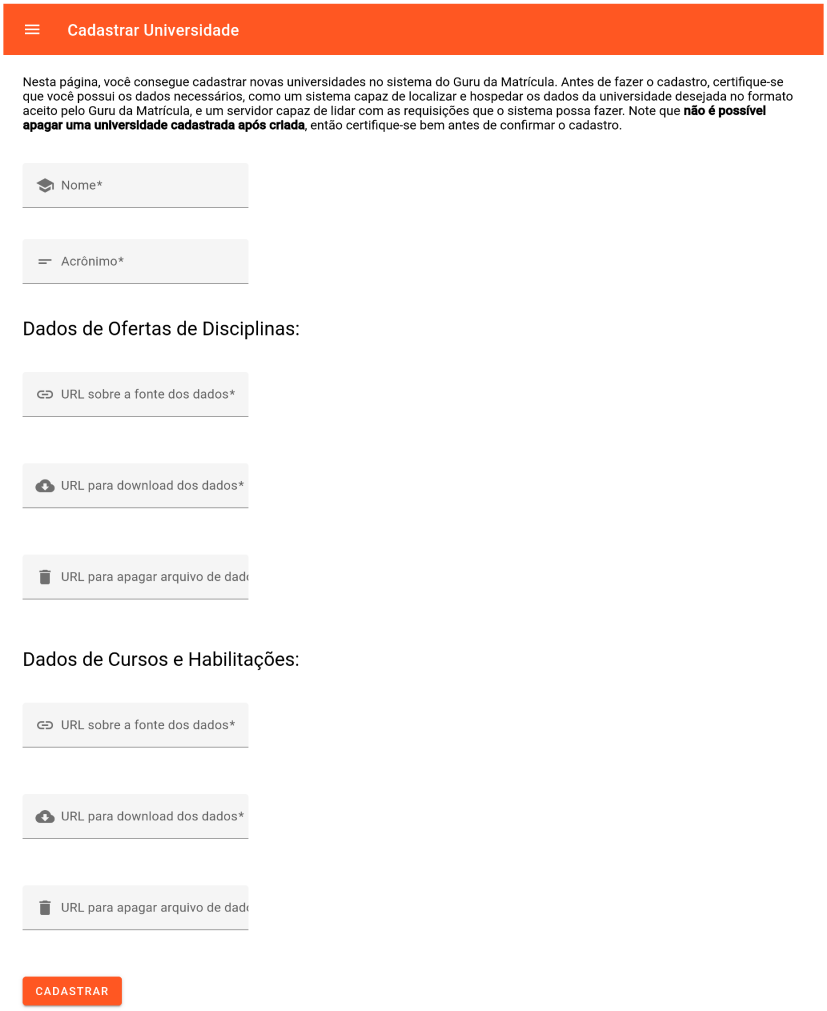

Figura 16. Página de cadastro de universidades.

bloco (3) da Figura 19, que são efeitos colaterais baseadas nas ações disparadas pelo usuário gerenciadas pela biblioteca Redux Saga[10]. Dentre os efeitos colaterais mais comuns, estão a comunicação com o servidor, tanto para carregamento quanto para escrita dos dados, e comunicação com outros componentes internos do *frontend*, como o combinador de turmas (bloco (4)) e o *Service Worker* (bloco (5)).

Essa comunicação com componentes internos, inclusive, acontece através do uso de mensagens, visto que tais componentes rodam em instâncias diferentes do motor Javascript no qual a página está sendo executada. Isso é feito tanto por questão da própria API do navegador, como no caso do *Service Worker*, onde apenas um *Service Worker* atende todas as instâncias da página aberta, como por questões de otimização, como no caso do combinador de turmas, que é executado em uma *thread* separada dentro do navegador para permitir a execução de tarefas pesadas computacionalmente (como encontrar combinações de turmas válidas, ou seja, sem conflitos e respeitando os filtros impostos pelo usuário, num conjunto com dezenas de milhares de combinações) sem afetar a ex-

[10] http://redux-saga.js.org

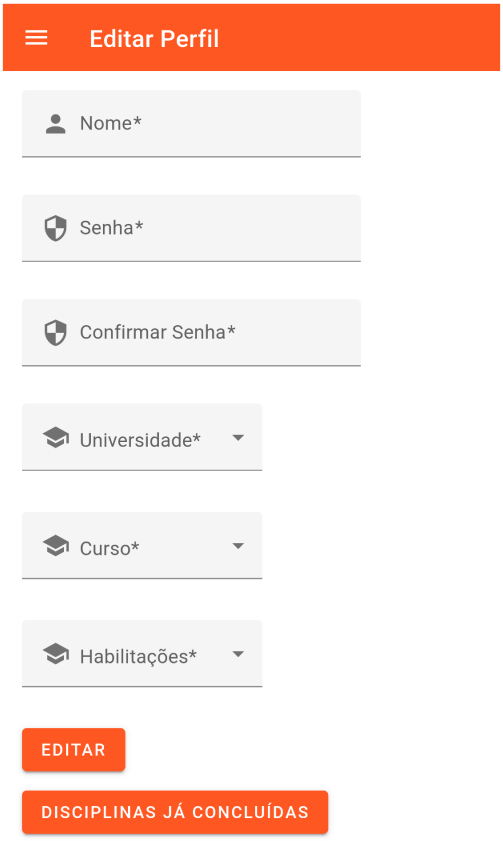

Figura 17. Página de atualização do perfil do usuário.

periência do usuário com a interface.

4.3.5. *Service Workers*

O *service worker* é uma espécie de *proxy* que reside entre a página que o usuário abriu e o servidor, para prover suporte a navegação *offline* do sistema. Dentre os demais componentes listados, é o único que não é crítico, ou seja, caso não seja carregado ou suportado, não tende a causar problemas para o uso **normal** da página, uma vez que tem atuação transparente sob o *Frontend*, fazendo com que apenas o recurso de navegação *offline* se torne indisponível.

O *service worker* trabalha armazenando e lendo dados salvos em um banco de dados disponibilizado por navegadores modernos acessível por uma API chamado *IndexedDB*. A implementação do *service worker*, conforme estrutura mostrada na Figura 20, é baseado na implementação de duas frentes de manipulação de dados: uma responsável por requisitar e armazenar os dados para navegação *offline* (bloco (6)), e outra para retornar tais dados da mesma forma que API *GraphQL* do *backend* (bloco (5)) faz (inclusive usando-se do mesmo modelo *GraphQL*), através do uso da implementação oficial do *GraphQL*, que permite que o *service worker* seja basicamente invísivel para o código executando na página do usuário, e intercepte as requisições sempre que possível.

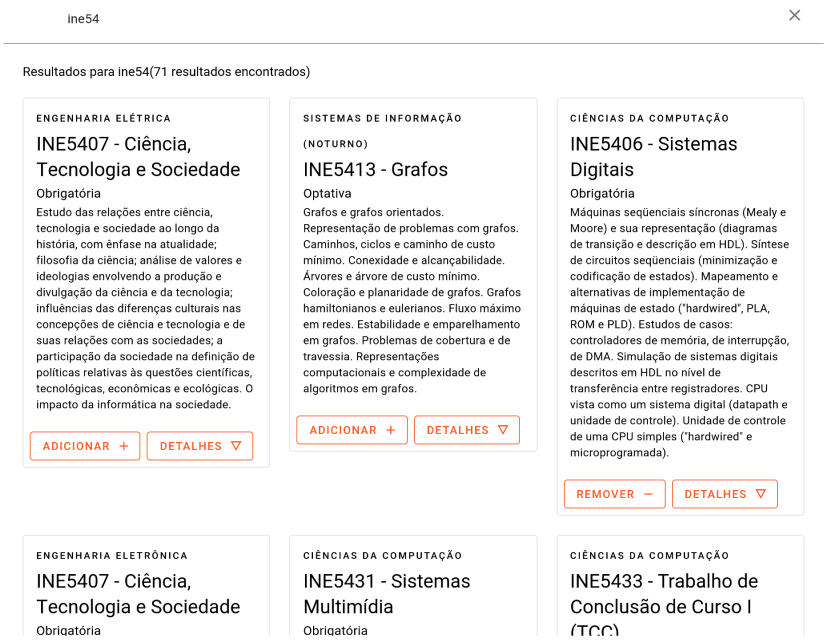

Figura 18. Página de seleção de disciplinas já feitas.

Além do armazenamento dos dados através da API *IndexedDB*, também são usados APIs disponíveis especificamente para *service workers* que permitem que os recursos estáticos da página, como código Javascript, *Hyper Text Markup Language* (HTML) e CSS, sejam armazenados de maneira persistente no *cache* do navegador, e requisitados de maneira apropriada, quase que totalmente gerenciada pelo próprio navegador, conforme representado no bloco (8) da Figura 20.

Deste modo, a sincronização dos dados é feita em duas partes distintas, uma para o *cache* dos recursos da página, que é gerenciado automaticamente pelo navegador ao carregar o *service worker* (usando o *cache* de recursos estáticos representados no bloco (8) da Figura 20), e outra, manual, em que é feita a atualização dos dados até então armazenados no *IndexedDB*, usando uma API do *backend* desenvolvida especificamente para sincronização de dados, representado pelo bloco (6).

Esta sincronização manual dos dados foi desenvolvida de forma a transferir a menor quantidade de dados possível com o servidor, mas ainda sendo leve tanto para o cliente quanto para o servidor. Para conseguir esse feito, o cliente envia uma lista composta por identificador e versão de cada tipo de entidade que está armazenado atualmente no banco de dados local do navegador, junto de dados básicos como universidade, período e curso de interesse, que devem ser manualmente escolhidos pelo usuário afim de reduzir tanto o espaço necessário no dispositivo para armazenamento dos dados quanto para reduzir a carga no servidor. Com base nesses dados, o servidor detecta e retorna uma lista composta por dados a inserir ou atualizar, e também uma lista composta por identificadores a serem deletados da base. Desta forma, o *service worker* consegue rapidamente detectar o conjunto mínimo de operações a serem feitas para ter a base de dados atualizada, maximizando a performance da operação.

Por fim, para auxiliar e facilitar algumas tarefas rotineiras (como inicialização e

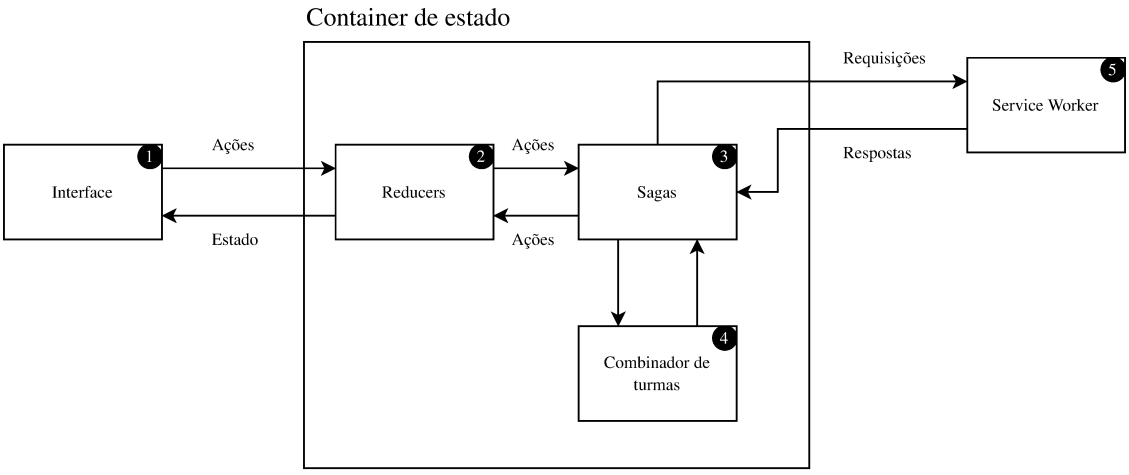

Figura 19. Estrutura geral do Container de estado.

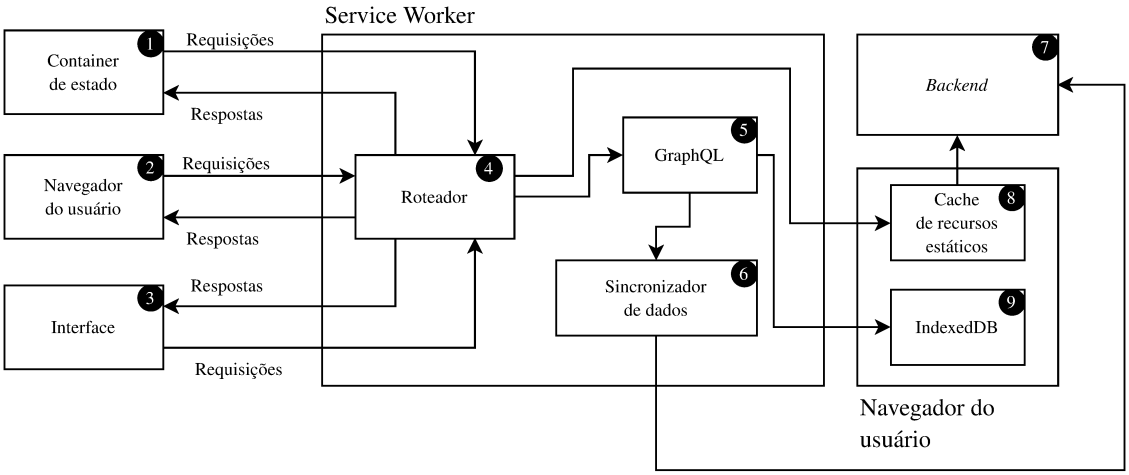

Figura 20. Estrutura geral do *Service Worker*.

uso do recurso de *cache* de arquivos estáticos do navegador, por exemplo) relacionadas à *service workers*, além de prover determinados recursos como o roteador (representado no bloco (4) da Figura 20), foi utilizado a biblioteca *Workbox*[11], desenvolvida pelo Google. É esse roteador que define tanto o que o *container de estado* (bloco (1)), o próprio navegador do usuário (bloco (2)) e a interface (bloco (3)) receberão como resposta ao fazer requisições para o servidor.

5. Estudos de Caso

Os dados armazenados e exibidos hoje no sistema são coletados a partir de duas universidades: UFSC e USP. Essas universidades foram escolhidas especialmente por publicarem os dados necessários na internet, de forma acessível à mecanismos automatizados para extração de dados. Entretanto, por usarem abordagens diferentes, foi necessário o desenvolvimento de mais de um mecanismo para realizar a coleta de dados e a exportação

[11] https://developers.google.com/web/tools/workbox

desses dados em um formato comum para importação pelo sistema, em cada um dos formatos necessários de acordo com o tipo de dado a ser extraído da universidade.

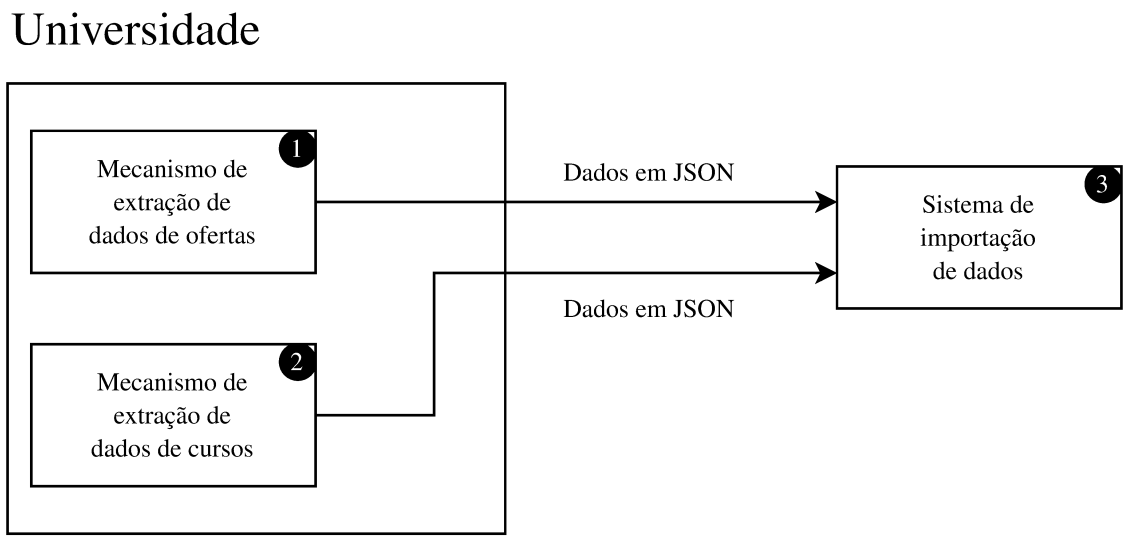

Figura 21. Estrutura interna da comunicação entre uma universidade e o sistema de importação de dados.

Desta forma, os dados de ofertas possuem um formato comum, em JSON, enquanto que os dados de cursos possuem outro formato comum, também em JSON. Esses formatos precisam ser exportados por cada universidade individualmente, ou extraídos por terceiros e expostos nos formatos necessários. A estrutura de funcionamento para uma universidade está representada na Figura 21, sendo que os mecanismos representados pelos blocos (1) e (2) são totalmente independentes do mecanismo representado pelo bloco (3) em sua implementação, e se comunicam com o esse último mecanismo apenas a partir do formato JSON comum (que é diferente entre dados de ofertas e dados de cursos).

5.1. UFSC

Os dados da UFSC são coletados a partir de duas fontes principais, disponibilizadas à partir do Sistema de Controle Acadêmico da Graduação (CAGR): o cadastro de turmas - de onde são coletados dados de oferta de turmas a cada semestre - e os curriculos dos cursos fornecidos no formato PDF, de onde são coletados dados detalhados sobre as disciplinas, pré-requisitos e suas características em cada curso disponível na universidade. Esses dados são processados por dois mecanismos totalmente independentes, que são capazes de lidar com as peculiaridades de cada formato, e exportam, cada um, um arquivo JSON que contem todos os dados relativos à informação solicitada, em um formato padronizado e aceito pelo sistema principal de importação.

5.1.1. Captura de dados de ofertas

O mecanismo de captura de ofertas de disciplinas foi desenvolvido em Go e funciona sob demanda e de forma majoritariamente distribuída, sendo capaz de capturar dados do cadastro de turmas do CAGR disponibilizado pela UFSC. Como esse cadastro de turmas

depende do uso de *HTTP cookies* para efetuar algumas funções (como paginação, por exemplo) corretamente, o sistema persiste essa informação entre cada acesso feito para cada conjunto de semestres e campus, e realiza tais acessos de maneira sequencial.

Dito isso, a primeira tarefa que o mecanismo realiza é acessar o cadastro de turmas da UFSC (bloco (1) da Figura 22) e identificar quais os semestres e campus disponíveis no sistema da universidade (bloco (2)). A partir daí, são disparadas tarefas, executadas paralelamente, para cada conjunto de semestre e campus (bloco (3)), e cada tarefa coleta os dados para esse conjunto de maneira sequencial, cada uma gerando um arquivo JSON. Quando todas as tarefas disparadas são terminadas, uma outra tarefa é agendada para concatenar todos os arquivos gerados para cada combinação de semestre e campus (bloco (4)). O arquivo JSON gerado pode, então, ser enviado para o sistema de importação de dados, representado no bloco (5).

Desta forma, ao término da execução, apenas um arquivo JSON é gerado, contendo todos os dados dos últimos semestres conforme configurado pelo usuário manualmente e seguindo o formato necessário para o processamento pelo sistema de importação de dados do sistema. O nome desse arquivo é, então, disponibilizado em local público e enviado a esse sistema de importação, que então realiza a importação dos dados capturados para o banco de dados. Note que, devido a essa manipulação dos arquivos JSON, é necessário que todas as unidades de execução de tarefas deste mecanismo possuam acesso a um mesmo armazenamento compartilhado, de forma que os arquivos possam ser lidos apropriadamente pelo concatenador de arquivos.

Dado a natureza do sistema da UFSC, foram adicionados algumas proteções contra problemas comuns ao capturar dados. Uma dessas proteções é a detecção de páginas duplicadas do cadastro de turmas, que ocorre através do armazenamento do *hash* de cada página retornada pelo sistema da UFSC, o que permite detectar erros relacionados à perda da sessão do sistema. Além disso, o Algoritmo é capaz de iniciar uma nova sessão automaticamente, de forma que a captura dos dados possam continuar caso este erro ocorra.

5.1.2. Captura de dados de cursos

Já o mecanismo de leitura dos currículos em formato *Portable Document Format* (PDF) não faz nenhuma requisição externa ao site da UFSC. Tal mecanismo é, na verdade, um

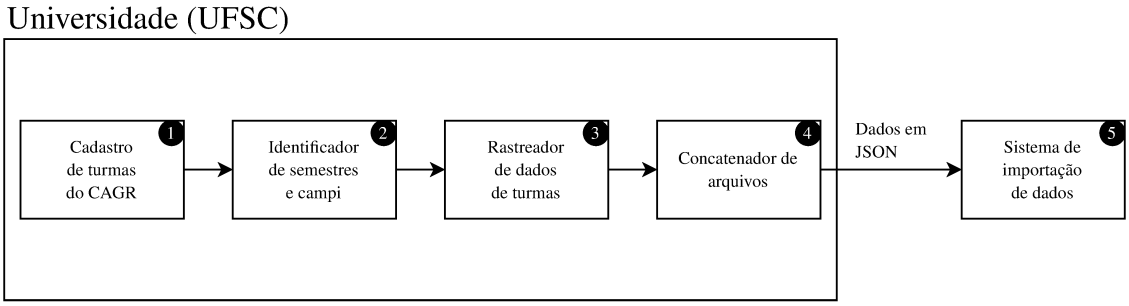

Figura 22. Estrutura interna do mecanismo de captura de dados de ofertas da UFSC.

programa, representado no bloco (2) da Figura 23 e desenvolvido em Java, que usa a biblioteca **Apache PDFBox** para ler o PDF de cada currículo (a partir de uma lista fornecida no bloco (1)), identificar os cursos, habilitações, disciplinas e demais dados dentro do arquivo e exportar, então um arquivo JSON contendo todas as informações identificadas no arquivo em questão, em um formato aceito pelo sistema de importação de dados do sistema (bloco (3)). Dentre as informações capturadas, se encontram:

- **Curso**: Código, Nome, Habilitações;
- **Habilitação**: Código, Nome, Fases;
- **Fase**: Nome, Disciplinas;
- **Disciplina**: ID,Código, Nome, Tipo, Descrição, Disciplinas relacionadas;

Esse sistema não é distribuído e foi desenvolvido desta forma devido ao fato de que esses dados não são atualizados com frequência, e a coleta de dados a partir de arquivos PDF é algo que ainda não foi implementado consistentemente em Go. Portanto, foi escolhido Java para esta tarefa especialmente devido à implementação do Apache PDFBox, que permitiu que tal implementação fosse possível.

Mesmo sendo desenvolvido em outra linguagem de programação, o fato do resultado do programa ser uma saída padronizada em JSON permite que o resto do sistema, desenvolvido em Go, consiga ler e importar os dados de maneira apropriada, permitindo aproveitar uma das vantagens em se ter um formato comum na comunicação entre sistemas desenvolvidos em diferentes linguagens.

5.2. USP

Os dados da USP são coletados a partir do Jupiter Web, a partir de duas implementações distribuídas em Go que acessam as páginas necessárias dentro do sistema do Jupiter Web (que é o sistema de graduação da USP) e identificam, a partir das páginas públicas desse sistema, todos os dados desejados por cada implementação, ou seja, os dados de ofertas de turmas para o semestre atual e também os dados relacionados ao currículo de cada curso que a universidade oferece. Ao final do processo, cada implementação gera um arquivo JSON único contendo os dados identificados no formato necessário para importação pelo robô do sistema.

As principais diferenças entre a UFSC e a USP no aspecto de coleta de dados ao fato de que a USP não depende de *HTTP Cookies* ou armazenamento de sessão para

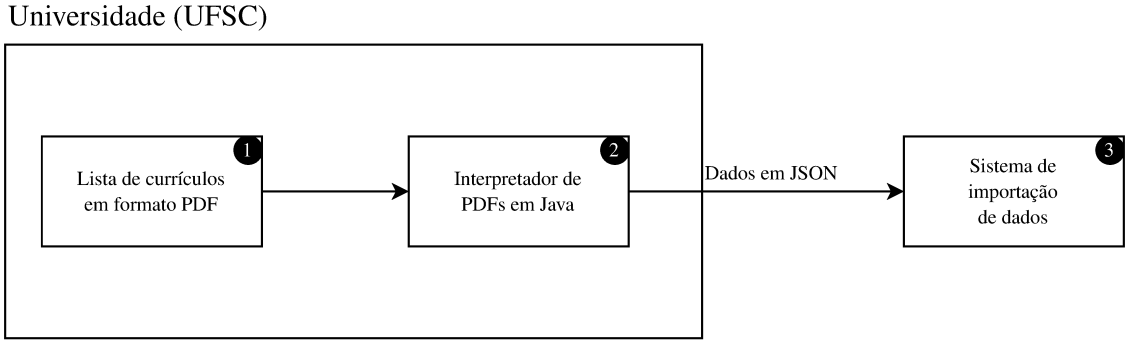

Figura 23. Estrutura interna do mecanismo de captura de dados de cursos da UFSC.

acesso aos dados, e, na USP, todos os dados são fornecidos em HTML, o que facilita consideravelmente a implementação do robô e evita a necessidade do uso de um programa em Java para realizar o processamento. Por outro lado, a quantidade de dados encontrado na USP é muito maior, dado que a universidade fornece, aos seus alunos, muito mais cursos e disciplinas do que a UFSC hoje fornece, o que faz com que a execução dos mecnaismos, mesmo que quase totalmente distribuída e paralela, seja mais demorada do que os mesmos mecanismos de coleta de dados para a UFSC.

5.2.1. Captura de dados de ofertas

O mecanismo de captura de ofertas de disciplinas da USP foi desenvolvido em Go e funciona, assim como o mecanismo de captura de dados de ofertas da UFSC, sob demanda e de forma distribuída. Pelo fato do Jupiter Web não depender de *HTTP Cookies*, uma grande parte do algoritmo é executada de maneira paralela, com apenas o balanceamento de tarefas - representado no bloco (3) da Figura 24 - e o concatenador de arquivos (bloco (5)) sendo etapas que não podem ser paralelizadas devido à seu funcionamento: no caso do balanceamento de tarefas, é necessário capturar a lista de campus e disciplinas a serem rastreadas (bloco (2)), a partir das páginas de turmas do Júpiter Web (bloco (1)), e então dividir igualmente o numero de tarefas a serem executadas de acordo com o numero de unidades de execução paralelas definido, enquanto que no caso do concatenador de arquivos todos os arquivos devem ser lidos, um a um, e seus valores escritos em um arquivo de saída, de maneira sequencial.

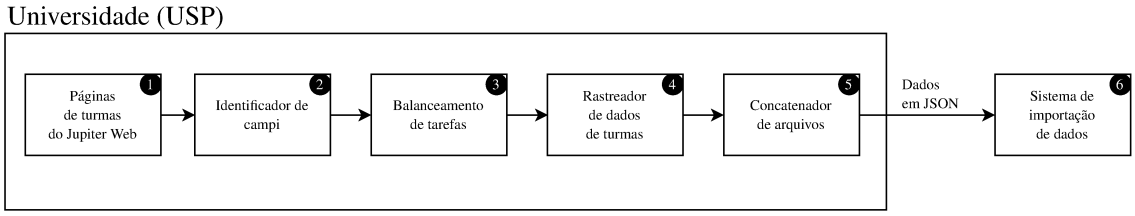

Figura 24. Estrutura interna do mecanismo de captura de dados de ofertas da USP.

Após tal balanceamento de tarefas, são disparadas tarefas que rastreiam, a partir do *parsing* do HTML da página, os dados de turmas para cada disciplina detectada, gerando arquivos JSON com os dados já coletados até então que contem um subconjunto das ofertas de turmas disponíbilizadas pela USP, correspondente às disciplinas acessadas. Isso é representado no bloco (4) da Figura 24.

Após a captura dos dados de ofertas e o término da execução de todas as unidades de execução paralelas, o concatenador de arquivos, representado no bloco (5) da Figura 24, é então executado, e recebe uma lista com os arquivos gerados e lê cada arquivo e passa os dados, ainda em formato JSON, para um único, novo, arquivo, que contem todos os dados capturados pelo mecanismo, e que é a saída final do mecanismo, no formato comum para ofertas aceito pelo robô de importação de dados, representado no bloco (6).

5.2.2. Captura de dados de cursos

O mecanismo de captura de dados de cursos da USP é desenvolvido em Go e é, também, majoritariamente paralelo, com apenas o balanceamento de tarefas e o concatenador de arquivos sendo etapas que não podem ser paralelizadas devido à seu funcionamento, conforme explicado na seção 5.2.1.

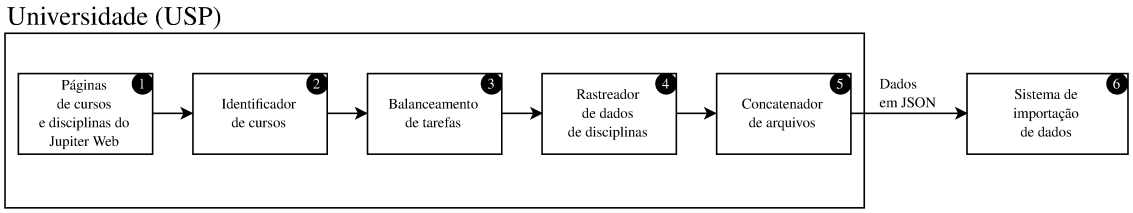

Figura 25. Estrutura interna do mecanismo de captura de dados de cursos da USP.

A captura dos dados de cursos acontece de forma parecida com a captura de dados de ofertas: o mecanismo realiza a identificação da lista de cursos (bloco (2) da Figura 25) a partir dos dados de cursos e disciplinas fornecidos pelo Jupiter Web ((1)), faz a divisão das tarefas pela quantidade de tarefas a serem executadas paralelamente (bloco (3)), rastreia os dados a partir do *parsing* do HTML das páginas e emite arquivos JSON contendo esses dados (bloco (4)), que são então concatenados por uma tarefa executada quando todo o rastreamento já foi concluído (bloco (5)), que gera, então, um único arquivo JSON contendo todos os dados capturados, que pode ser enviado para o sistema de importação de dados (bloco (6)).

Uma característica importante deste mecanismo é que a captura de detalhes (tais como descrição da disciplina, por exemplo) das disciplinas é **opcional**. Isso acontece pois, ao fazer a captura desses dados, o processo de rastreamento se torna muito mais lento, visto que o mecanismo precisa acessar a página de cada disciplina individualmente para obter tais informações, e tal procedimento não pode ser balanceado para processamento paralelo devido ao fato desses dados estarem diretamente associados com cada curso , com o processo de balanceamento inicialmente realizado não sendo tão eficiente nesse caso.

6. Experimentos

Durante o desenvolvimento das diferentes partes do sistema, ao realizar testes de funcionamento, foi possível verificar diferentes dados que podem ser analisados para permitir, a usuários que não participaram do desenvolvimento do sistema, uma breve noção dos diferentes desafios envolvidos durante esse processo. Muitos desses dados são, de fato, "invísiveis" para o usuário final, que vai usar o sistema para simular sua matrícula, entretanto, mesmo esses dados possuem importância fundamental na viabilidade do mecanismo.

Um exemplo básico de dado desse tipo é o consumo de memória *Random Access Memory* (RAM) durante a importação de dados: computadores não possuem memória ilimitada, portanto, é desejado que o sistema possua baixo consumo de memória de forma

a possibilitar tanto a manutenção da *performance* do mecanismo para todos como também permitir que mesmo universidades tão grandes quanto a USP, por exemplo, possam ter seus dados importados e fornecidos dentro do mecanismo sem exigir um computador com muita memória RAM, e portanto mais caro, para que isso seja possível.

Entretanto, existem dados que afetam diretamente o usuário. Um exemplo disso é o tamanho dos recursos que compõem o *frontend*: tais recursos são enviados diretamente para o navegador do usuário e, mesmo que possam ser compactados (reduzindo a quantidade de dados consumido na franquia móvel, por exemplo), ainda precisam ser executados no dispositivo local, do usuário, o que implica numa possível lentidão, e até travamentos caso o dispositivo do usuário não tenha capacidade de processamento suficiente para lidar com o código da página.

Por isso, a análise de dados, envolvendo diferentes experimentos, envolverá três partes principais: uma para os coletores de dados, apresentados no Seção 5, uma para o *backend*, apresentado na Seção 4.2, e, por fim, uma para o *frontend*, apresentado na Seção 4.3.

Um aspecto importante a ser considerado em relação aos testes de uso de memória apresentados é que esses testes considerarão, apenas, o *Unique Set Size* (USS), que é a quantidade de memória que seria liberado se o processo fosse terminado nesse exato instante. Isso é feito pois, dependendo de fatores como o banco de dados usado, por exemplo, o uso de memória do aplicativo pode ser maior em decorrência do uso de bibliotecas compartilhadas com outros aplicativos, e também com o mapeamento de arquivos em memória (cuja quantidade de dados em memória é administrado completamente pelo sistema operacional).

Todos os testes executarão em um computador com Ryzen 7 1700, 16GB de RAM e SSD 1TB NVME, mas, mesmo com essa configuração, todos os testes foram feitos usando apenas 4 unidades de execução, visando reduzir memória e mostrar condições "ideais" de uso. Além disso, o banco de dados usado é o *Bolt*, sem *cache*, e com o uso do codificador *Gob* sem compressão para codificação dos dados. Por fim, o programa foi compilado usando *Go 1.13* e executado em sistema operacional Antergos Linux.

6.1. Coletores de dados

Os coletores de dados desenvolvidos para a UFSC e para a USP são ferramentas que a princípio não precisam rodar no mesmo servidor que o resto do sistema, mas que lidam com uma enorme quantidade de informações ao depender, basicamente, do *parsing* de páginas HTML fornecidas pelas universidades. Isso acontece pois tais sistemas precisam acessar uma quantidade muito maior de páginas, e interpretar muito mais conteúdo, para poder extrair apenas os dados que interessam.

Devido a isso, é necessário que esses coletores sejam otimizados para consumir pouca memória mesmo ao acessar muitas páginas. Isso é feito principalmente com o uso de técnicas de *parsing* iterativo, que permitem ir descartando todo o conteúdo da página que não é interessante para extração a partir do momento em que esse conteúdo é encontrado. Além disso, mesmo o conteúdo já extraído não é armazenado em grandes quantidades na memória, mas imediatamente codificado e transferido para o armazenamento persistente, o que facilita tanto na recuperação de erros quanto na diminuição do consumo de memória.

6.1.1. UFSC

No caso da UFSC, a extração de dados mais complexa acontece na extração de dados de cursos, que precisam lidar com o formato PDF, que acaba exigindo o uso de uma biblioteca Java para fazer a interpretação do conteúdo, e a construção de grafos complexos para cada curso, com a necessidade de armazenar os dados de cada disciplina em memória durante o processamento do curso, isso acaba exigindo maior consumo de memória, e, devido ao *parsing* do PDF, também de processador.

Em relação ao extrator de dados de turmas da UFSC, o maior desafio é o armazenamento da sessão (salvos em HTTP *cookies*), mas isso não implica em maior consumo de memória nem processamento visto que a quantidade de dados que é necessário armazenar é pequena. Além disso, como não há a necessidade de criar um grafo baseado na relação entre as disciplinas, é possível detectar e persistir os dados de turma coletados diretamente para o disco, reduzindo o consumo de memória.

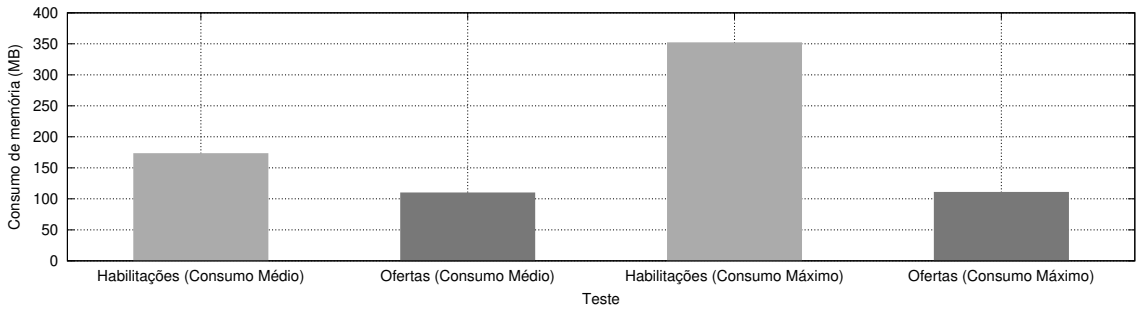

Figura 26. Comparação de consumo de memória entre os coletores de dados da UFSC.

O uso de memória dos coletores da UFSC, como já antecipado, foi maior no extrator de dados de cursos, devido ao fato de ser desenvolvido em Java e precisar lidar com o *parsing* de arquivos PDF, incluindo o armazenamento temporário de todas as disciplinas de curso e suas devidas relações. É possível observar esas diferença no consumo de memória entre os coletores de dados da UFSC na Figura 26.

Um aspecto importante que vale destacar nessa comparação é que, a princípio, o coletor de dados de habilitações compreende apenas a parte de interpretação dos arquivos PDF, sem, portanto, considerar *download* e rastreamento desses arquivos.

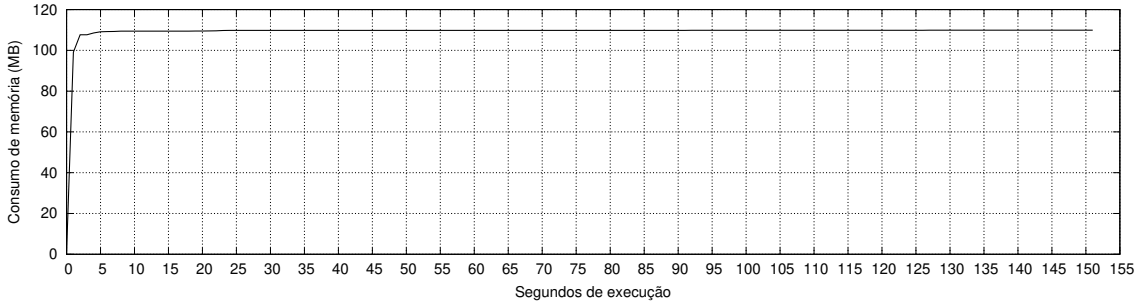

Figura 27. Registro de consumo de memória durante a execução do coletor de dados de turmas dos últimos 3 semestres da UFSC.

Por fim, foi possível observar, através da Figura 27, que o consumo de memória do coletor de dados de turmas da UFSC possui uso de memória razoavelmente constante, enquanto que o coletor de dados de cursos/habilitações quase que continuamente aumenta o consumo de memória durante a execução, como é possível visualizar na Figura 28, com quedas em alguns momentos devido ao próprio *Garbage Collector* do Java.

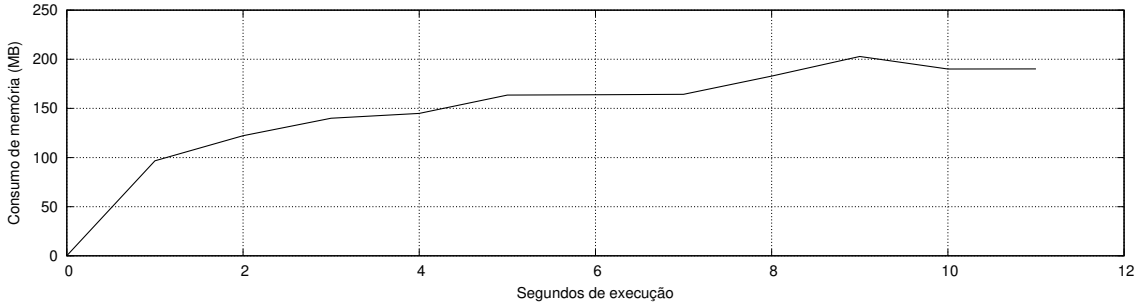

Figura 28. Registro de consumo de memória durante a execução do coletor de dados de cursos/habilitações da UFSC.

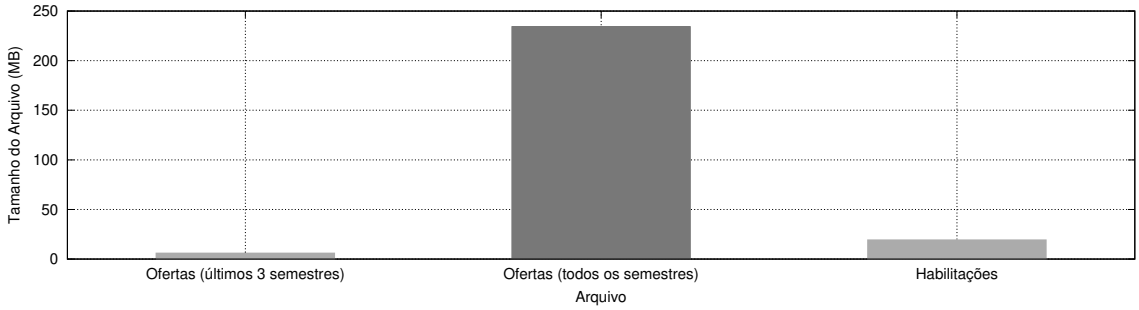

Figura 29. Tamanho dos arquivos gerados durante a execução dos coletores de dados da UFSC.

Em relação ao tamanho dos arquivos gerados com os dados da UFSC, mostrados na Figura 29, foi possível observar que, salvo nos casos onde os dados de turmas de todos os semestres da universidade foram coletados, o tamanho do arquivo gerado pelo extrador de cursos foi sempre maior que os dados de turmas dos semestres recentes coletados. Isso acontece pois a UFSC tem atualmente cerca de 127 cursos, distribuidos por 279 habilitações, com um total de 19806 disciplinas, mas a maior parte dessas não possui ofertas semestralmente: Em 2019-2, por exemplo, a UFSC ofereceu 6613 turmas, distribuídas por apenas 3741 disciplinas.

Isso faz com que, ao coletar 3 semestres de dados de ofertas de turmas (o que corresponde a 1 ano para a UFSC), o arquivo com esses dados fique com menos da metade do tamanho do arquivo com os dados de cursos, como é possível visualizar na Figura 29.

Por fim, como é possível reparar na Figura, os dados podem ser compactados e, ao fazer isso, o tamanho dos arquivos diminui drasticamente, como mostrado na Figura 30, ajudando a reduzir a necessidade de um grande espaço de armazenamento para os dados, além de não perder a compatibilidade com o resto do sistema de importação (caso a ferramenta de compactação seja feita em formato *GZIP*, que é suportado nativamente desde que configurado corretamente).

No caso da duração para coleta dos dados da UFSC, foi observado que o coletor de dados de habilitações/cursos é muito mais rápido que o coletor de dados de turmas, como mostrado na Figura 31. Isso acontece pois o coletor de dados de turmas, além de fazer o *parsing* e extração dos dados a partir das páginas do CAGR, também precisa acessar cada página individualmente, enquanto que o coletor de dados de habilitações/cursos apenas faz o *parsing* a partir dos arquivos PDF locais dos currículos de cursos da UFSC, que o usuário deve baixar por conta própria.

Essa diferença de abordagem já é o suficiente pra causar tal vantagem nos tempos de execução em comparação com a coleta de dados de turmas da UFSC, uma vez que, embora o download das páginas por si só seja lento, está sujeito a limitações do próprio CAGR, como o fato de não poder fazer as requisições de maneira totalmente paralelizada, e também a necessidade de colocar um intervalo entre cada requisição para que o servidor da universidade não fique sobrecarregado com o excesso de conexões, levando ao eventual bloqueio de acesso ás páginas.

6.1.2. USP

Em relação à USP, o maior desafio que os coletores de dados enfrentam é lidar com a quantidade de dados que a universidade disponibiliza, especialmente quando comparado com a UFSC. Essa quantidade de dados leva a desafios tanto em termos de armazenamento, pois o arquivo final com os dados tende a ficar muito grande, quanto ao consumo de memória, que pode crescer absurdamente caso o algoritmo não seja cuidadosamente planejado.

Outro aspecto que chama atenção, também, é o fato de que devido ao tamanho da universidade e a própria estrutura do site da USP, se torna necessário acessar uma quantidade enorme de paǵinas, o que faz com que a execução dos coletores de dados acabe ficando muito mais lenta caso o sistema não seja apropriadamente desenvolvido para executar de maneira paralela, dividindo, para várias unidades de execução, o trabalho de acessar e extrair dados das páginas. Assim como no caso da UFSC, no entanto, é necessário limitar a velocidade com que as páginas são acessadas, de forma que não sobrecarregue os servidores da universidade e leve ao bloqueio das páginas com os dados para o público.

Devido ao fato dos coletores de dados da USP serem desenvolvidos em Go, foi

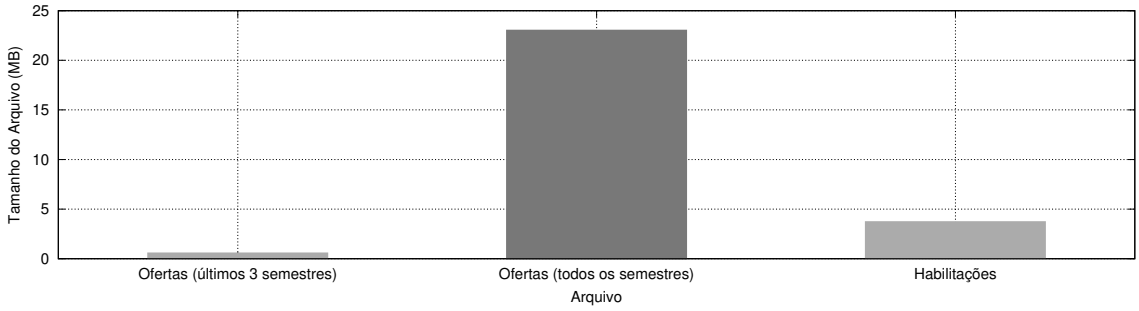

Figura 30. Tamanho dos arquivos gerados durante a execução dos coletores de dados da UFSC, compactados usando *GZIP*.

possível observar um baixo consumo de memória, como mostrado na Figura 32, muito menor do que o coletor de dados de cursos da UFSC, que foi desenvolvido em Java.

Esse consumo da memória, assim como o coletor de dados de turmas da UFSC, é praticamente constante, com uma característica importante relacionada à própria linguagem se destacando em ambos os coletores de dados: O fato de que, após alguns minutos, ou até mesmo horas, de execução, o consumo de memória cai drasticamente, em decorrência da ação do *garbage collector* da linguagem, como é possível visualizar tanto na Figura 33 como na Figura 34.

Durante a maior parte da execução do programa, entretanto, foi possível observar um crescimento extremamente pequeno da quantidade de memória alocada. Isso se deve à priorização do Go, com o auxílio do próprio sistema (através do uso intensivo de reciclagem de memória), em reusar blocos de memória já alocados em vez de constantemente liberar e alocar novos blocos, o que possibilita maior desempenho do sistema. Em alguns casos, entretanto, foi possível observar quedas no consumo de memória apenas quando o algoritmo já estava quase no fim de sua execução, devido à diminuição na carga de trabalho em decorrência de um balanceamento de tarefas não-perfeito, que faz com que determinadas unidades de execução fiquem sem tarefas para fazer e resulta na diminuição do uso da memória do programa, como mostrado na Figura 34.

Já no caso do tamanho dos arquivos gerados da USP, foi possível notar que o tamanho do arquivo com os dados de cursos é muito maior que o tamanho do arquivo de dados de turmas, como mostrado na Figura 35. Isso acontece por um motivo similar ao da UFSC, mas em uma escala muito maior, devido ao fato de que a USP fornece apenas o último semestre de dados de turmas (diferente da UFSC, onde é possível definir o número de semestres a ser capturado), e também devido ao fato de que a USP possui um número muito maior de cursos e de turmas: são 131 cursos, distribuídos por 765 habilitações e 57408 disciplinas. Em relação aos dados de ofertas, para um único semestre, foram detectados 11391 turmas distribuídas por 5638 disciplinas, o que acaba por justificar o tamanho maior do arquivo quando comparado com a UFSC, por exemplo.

Quando esses arquivos são compactados usando *GZIP*, conforme mostrado na Figura 36, a redução de espaço consumido em relação aos arquivos não-compactados é maior do que no caso da UFSC, devido ao fato de que, por possuir mais dados e, naturalmente, muitos desses dados acabarem sendo repetidos, é possível que o algoritmo de compressão possa compactar melhor os dados. Entretanto, mesmo com esse fator, foi possível observar que o arquivo contendo os dados de habilitações/cursos ainda ficou

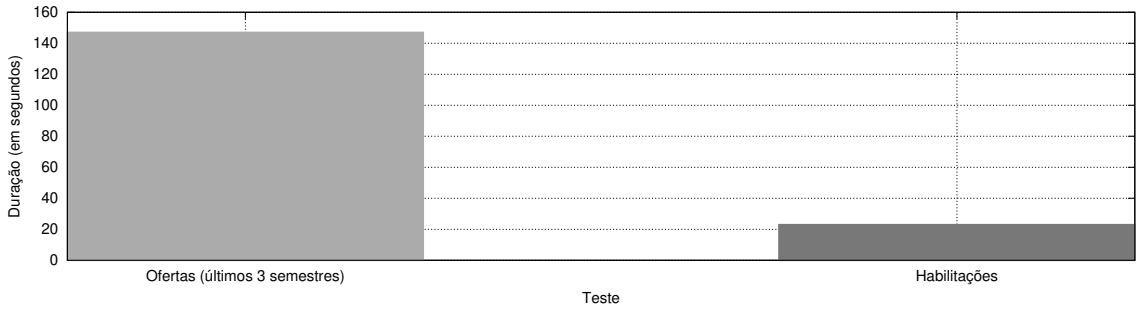

Figura 31. Duração da execução das coletas de dados da UFSC.

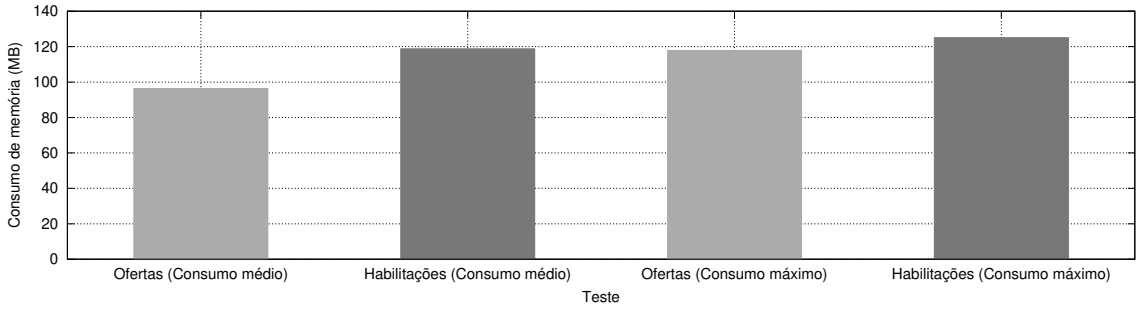

Figura 32. Comparação de consumo de memória entre os coletores de dados da USP.

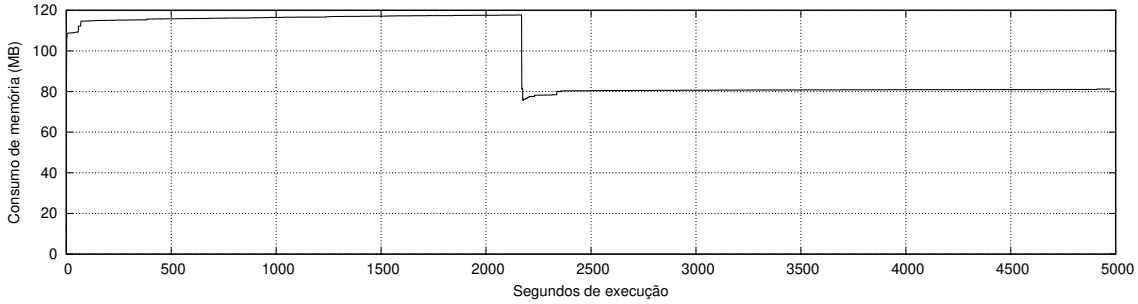

Figura 33. Registro de consumo de memória durante a execução do coletor de dados de turmas da USP.

grande, ocupando 77MB mesmo após a compactação com *GZIP*.

Em relação à duração da coleta de dados da USP, foi possível observar que, diferente da duração da coleta de dados da UFSC, o coletor de dados de habilitações/cursos demorou muito mais para executar do que o coletor de dados de ofertas, como mostrado na Figura 37. Isso se deve, principalmente, ao fato de que o coletor de dados de habilitações/cursos da USP precisa acessar uma quantidade enorme de páginas, afim de capturar os dados de todas as disciplinas que a universidade oferece, além do fato de que, diferente do caso da UFSC, a USP não fornece páginas que detalham todas as informações que o coletor de dados procura coletar, como, por exemplo, descrições da disciplina, o que faz com que seja necessário acessar uma página para cada disciplina afim de obter tais informações.

Devido a isso, o sistema precisa acessar, sequencialmente, as páginas de cada disciplina, para cada curso encontrado, o que aumenta e muito o tempo necessário para capturar todos os dados de um curso, por exemplo. Além disso, a USP também fornece dados de versões anteriores dos cursos, que também são coletados, o que acaba por aumentar muito o número de disciplinas encontrados e, portanto, o tempo necessário para coletar todos os dados.

Enquanto isso, no caso do coletor de dados de ofertas, o sistema acaba capturando apenas os dados do último semestre disponibilizados pela universidade, e consegue paralelizar muito bem o acesso às páginas para coleta de informação entre as diferentes unidades de execução, o que permite que o sistema como um todo seja mais rápido em relação à coleta de dados de habilitações/cursos, mas, devido ao tamanho da universidade,

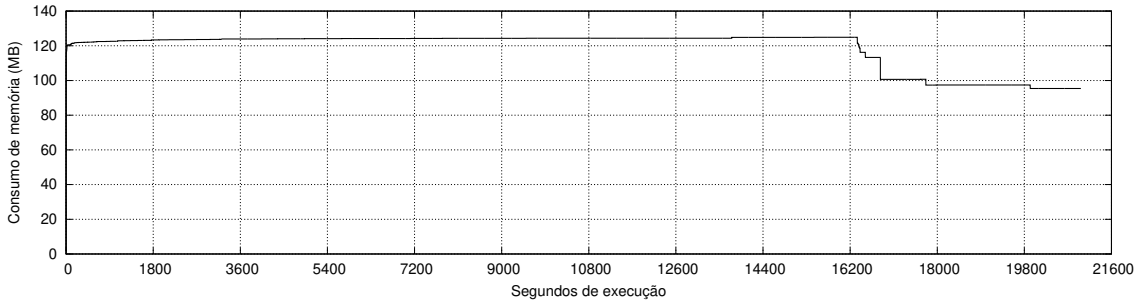

Figura 34. Registro de consumo de memória durante a execução do coletor de dados de habilitações/cursos da USP.

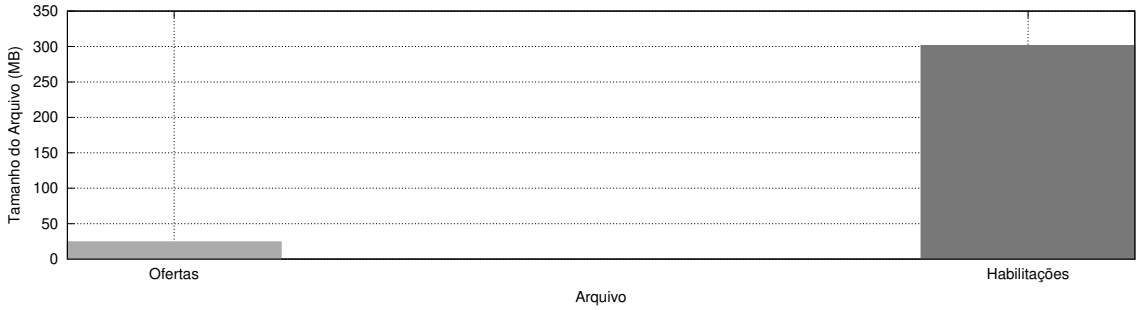

Figura 35. Tamanho dos arquivos gerados durante a execução dos coletores de dados da USP.

ainda seja mais lento do que a coleta de dados de ofertas da UFSC, por exemplo.

6.2. Backend

O *backend* do sistema é responsável por importar os dados das universidades e, além disso, atender as requisições dos usuários. Devido a isso, é a parte que precisa ser mais otimizada no sistema, pois precisa lidar com uma grande quantidade de dados, tanto em termos de processamento quanto em termos de gravação (durante o processo de importação) e leitura (durante as requisições do usuário).

O momento mais desafiador, nesse caso, é durante o processo de importação, uma vez que o sistema precisa ler os arquivos JSON (que podem ser grandes,a ponto de possuir mais de 100MB no caso da USP, como mostrado na seção anterior), fazer a comparação dos dados com o arquivo mais recentemente processado e então importar esse arquivo paralelamente, tanto diretamente para o banco de dados quanto para o índice. Como os arquivos são grandes, importar esses arquivos exigem que o algoritmo não carregue todo o arquivo em memória, caso contrário o consumo de memória cresceria de acordo com o tamanho dos arquivos, o que prejudicaria toda o desempenho do sistema de acordo com o tamanho do sistema.

Um detalhe importante sobre esse sistema é que as diferentes configurações de *cache* e banco de dados podem influenciar diretamente algumas dessas métricas, devido a características próprias desses subsistemas. Um exemplo disso é o fato de que o uso do banco de dados *Bolt*, por exemplo, pode aumentar bastante algumas métricas de consumo de memória devido ao uso de arquivos mapeados em memória, que acabam fazendo com que o aplicativo acabe "consumindo" mais memória que, na verdade, é automaticamente

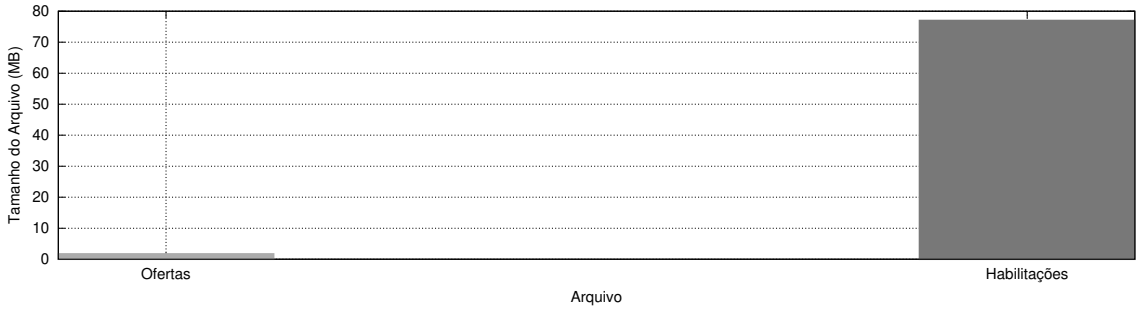

Figura 36. Tamanho dos arquivos gerados durante a execução dos coletores de dados da USP, compactados usando *GZIP*.

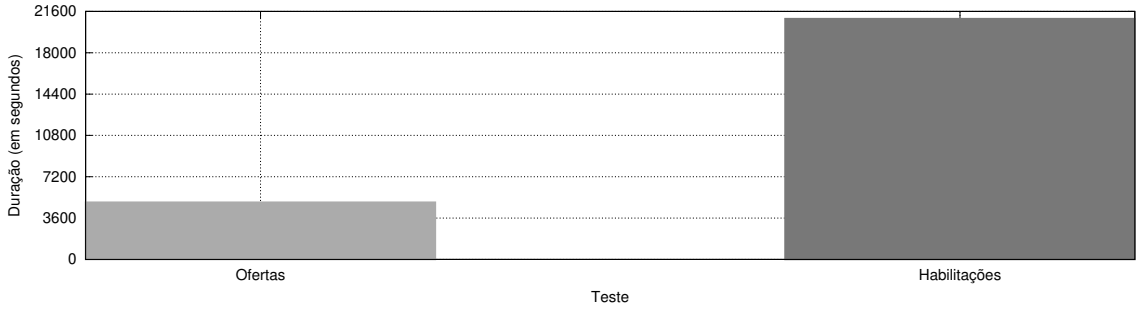

Figura 37. Duração da execução das coletas de dados da USP.

gerenciada pelo sistema operacional e na tentativa de otimizar ao máximo o acesso ao arquivo de banco de dados de acordo com a quantidade de memória disponível.

Além disso, é importante observar que todas as análises foram feitas com o banco de dados inicialmente limpo, com apenas um usuário e as duas universidades correspondentes cadastradas, mas sem a presença de nenhum dado de turma ou de curso, de forma a facilitar a reprodução dos experimentos e também permitir uma análise melhor sobre o impacto da importação de cada conjunto de dados.

6.2.1. Uso de memória durante a importação de dados

Em relação ao uso de memória do sistema de importação de dados, mostrado na Figura 38 e na Figura 39, foi possível observar que o consumo depende, em parte, ao tamanho do arquivo a ser importado e aos tipos de dados armazenados. Durante a análise, foi possível notar que os dados da USP consomem um pouco mais de memória, mas não foi possível visualizar um crescimento linear quando em comparação com os dados de consumo de memória registrados para importação dos dados da UFSC, mostrando que não há uma relação estrita entre tamanho do arquivo e consumo de memória.

Durante o processamento, foi possível observar um pico no consumo de memória de até 379MB, decorrente da presença de objetos grandes na memória e também ao fato de que o *runtime* Go sempre aloca memória adicional afim de evitar chamadas recorrentes ao sistema operacional. É possível observar os picos de memórias correspondentes na Figura 39.

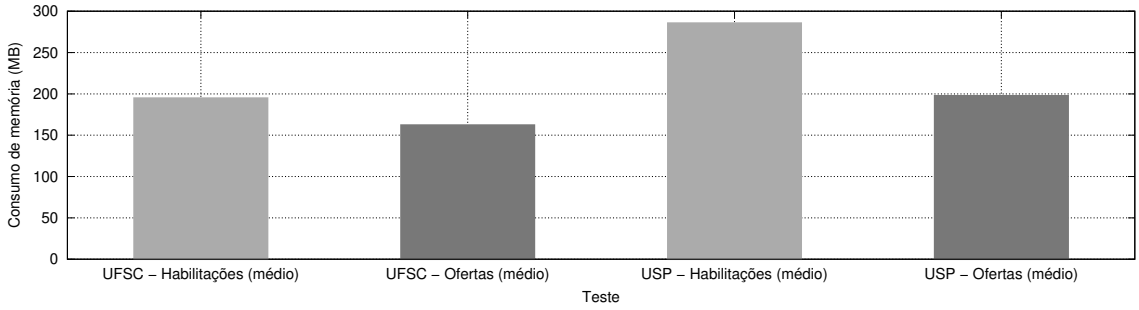

Figura 38. Consumo de memória médio do sistema de importação de dados.

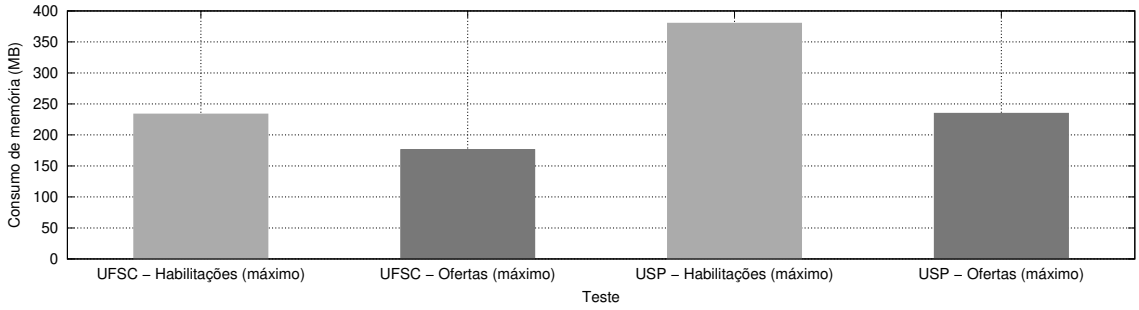

Figura 39. Consumo de memória máximo do sistema de importação de dados.

Também foi possível observar, ao comparar o consumo médio com o consumo máximo, que, devido à longa duração da importação, no geral o consumo se mantem relativamente baixo, aumentando apenas em momentos mais críticos como a criação do índice de busca, que exige o armazenamento de mais dados temporariamente em memória para permitir a criação do índice reverso de modo otimizado.

6.2.2. Duração da importação

Sobre a duração de importação, apresentado na Figura 40, assim como no caso do consumo de memória da importação, foi possível observar que o tempo cresce de acordo com o tamanho do arquivo a ser importado, uma vez que a quantidade de registros cresce de acordo com o tamanho do arquivo e, também, não é possível paralelizar infinitamente a

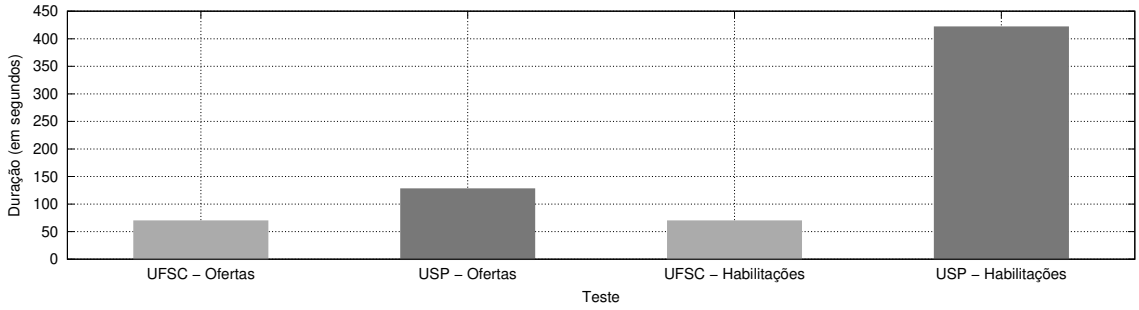

Figura 40. Comparação da duração da importação de diferentes conjuntos de dados no sistema.

carga de trabalho.

Entretanto, assim como acontece com o caso do consumo de memória, o crescimento observado não foi linear, ou seja, não existe uma relação direta entre o número de registros de uma universidade e a duração da importação.

Um aspecto importante sobre as medições relacionadas à duração é que, pelo fato do sistema como um todo ser um servidor desenvolvido com a ideia de não ter um final específico de execução, a medição de duração do processo de importação ocorreu numa tarefa de monitoramento executada periodicamente e que é capaz de determinar se a importação acabou ou não. Essa tarefa periódica é executada a cada 1 minuto e, para efeitos de medição, foi adicionada uma condição onde ao detectar o término da importação o sistema encerra todo o servidor após um intervalo de 10 segundos. Ou seja, a margem de erro dos dados coletados é de, no máximo, 1 minuto e 10 segundos, no pior caso, mas ainda corresponde adequadamente à duração do processo de importação em um ambiente real dado a própria totalmente assíncrona natureza do sistema.

6.2.3. Tamanho do banco de dados

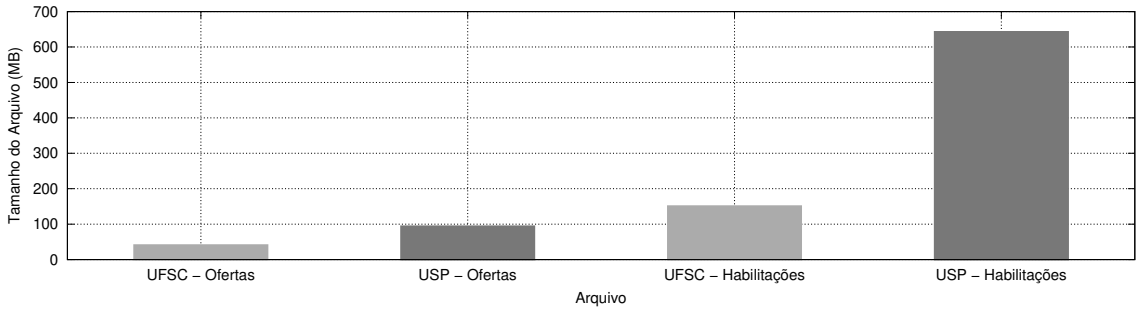

Figura 41. Comparação do tamanho do banco de dados após a importação de diferentes conjuntos de dados no sistema.

Em relação ao tamanho do banco de dados, mostrado na Figura 41, foi possível notar uma relação entre o tamanho do arquivo de entrada e o tamanho final do banco de dados, que aumenta numa escala maior que no caso da memória e da duração da importação. Isso é natural e esperado, devido ao fato de que todos os dados do arquivo de entrada são salvos e indexados de maneira à permitir o acesso e atualização mais eficiente possível aos dados, o que implica no uso de um maior consumo de espaço em disco conforme a quantidade de dados aumenta.

Graças à isso, tanto o banco de dados de turmas quanto a de habilitações para os dados da UFSC acaba por ser muito menor que o banco de dados correspondente para a USP, visto que esta última possui uma quantidade muito maior de dados que a primeira, tanto em dados de turmas quanto de habilitações.

Além disso, no banco de dados também são armazenados dados relacionados à execução das tarefas de importação, afim de permitir o gerenciamento das tarefas de maneira confiável. Isso significa que o tamanho do banco de dados acaba aumentando conforme a tarefa precisa de mais tarefas para sincronização da execução, mas essa parcela

ainda é menor do que o espaço necessário para armazenamento dos outros dados, como as tabelas próprias com todos os dados e também os índices de busca.

6.3. *Frontend*

O *frontend* do sistema é a parte responsável por servir de interface entre o usuário e o resto do sistema, e que é executado diretamente no dispositivo do usuário. Graças a isso, é necessário que um cuidado especial na otimização e na quantidade de código a ser enviado para usuário para que a maior variedade possível de dispositivos seja suportado e, devido a esse, o sistema seja rápido de carregar e usar, mesmo em redes móveis de internet.

Devido a essas características, presentes em todo e qualquer *frontend*, surgiram diferentes metodologias de análise para determinar as propriedades dos sistemas e mensurar a experiência do usuário logo ao carregar a página, que é um dos momentos mais críticos, pois é quando há maior chance do usuário desistir de acessar a página: segundo um estudo [12] do Google, por exemplo, cerca de 53% dos usuários desistem de acessar uma página que demorou mais de três segundos para carregar.

Graças a isso, serão feitas três análises distintas em relação ao *frontend*: uma de tamanho dos arquivos, analisados individualmente e também no contexto de uma página de plano, uma análise com o *Lighthouse*, que foi desenvolvido justamente para analisar os diferentes aspectos que podem afetar a experiência do usuário em páginas *web*, como desempenho, acessibilidade e *Search Engine Optimization* (SEO), por exemplo.

6.3.1. Tamanho dos arquivos

Em relação ao tamanho dos arquivos que compõem o *frontend*, foi possível observar a existência de arquivos relativamente grandes, devido principalmente ao uso de bibliotecas Javascript relativamente grandes por todo o site, como o *React*. Os dados de nome e tamanho do arquivo estão apresentados na Tabela 1.

Quem faz o gerenciamento em baixo nível dessa lista de arquivos é, a princípio, o *Webpack*[13], que recebe como entrada o arquivo com código fonte e aplica diferentes transformações para que esse código possa ser executado pelo navegador do usuário. Dentre essas transformações, está a detecção de recursos comuns para todos os arquivos, que são então colocados no *main.js* e carregados em todas as páginas. Neste caso, exemplos de recursos comuns seriam casos como bibliotecas, como *React* e *RMWC*, por exemplo, assim como componentes que são acessados em todas as páginas.

Já os outros arquivos são menores pois, a princípio, são arquivos projetados para serem carregados sob demanda, quando o usuário navega por essas outras páginas, e possuem apenas os componentes, bibliotecas e demais recursos necessários por essas páginas, já ignorando tudo o que já está presente no arquivo principal. Desta forma, bibliotecas grandes como *React* acabam por ser carregadas apenas uma vez, o que ajuda a melhorar a eficiência do *frontend* como um todo e também diminui a quantidade de dados transferida entre o servidor e o dispositivo do usuário.

[12] https://www.thinkwithgoogle.com/marketing-resources/

Nome	Tamanho
main.js	341KB
main.css	159KB
0.bundle.js	68KB
1.bundle.js	14KB
2.bundle.js	32KB
3.bundle.js	11KB
4.bundle.js	15KB
6.bundle.js	61KB
7.bundle.js	29KB
8.bundle.js	37KB
9.bundle.js	15KB
10.bundle.js	23KB
11.bundle.js	16KB
12.bundle.js	19KB
13.bundle.js	12KB
14.bundle.js	17KB
15.bundle.js	14KB
16.bundle.js	9.9KB
17.bundle.js	8.2KB
18.bundle.js	9.5KB
20.bundle.js	48KB
worker.js	7.7KB
sw.js	162KB

Tabela 1. Nome e tamanho de cada arquivo gerado para o *frontend*. Com resultado já minimificado, mas sem compressão.

Um aspecto importante dos arquivos que constituem o *frontend* é que, devido ao fato dos *scripts* serem código que é executado no próprio navegador do usuário, tal código pode ser não só minimificado como também compactado usando algoritmos de compressão que são suportados nativamente pelo navegador, como *GZIP* e *Brotli*. O impacto desses algoritmos de compressão nos arquivos do *frontend* pode ser visualizado na Figura 42, que mostra o tamanho do maior arquivo do *frontend* (já minimificado) após a aplicação de cada algoritmo de compressão.

6.3.2. Análise com Lighthouse

A partir da análise do *frontend* feita com a ferramenta *Lighthouse*, foi possível reparar que, por sua própria natureza moderna, a aplicação já cumpre boa parte dos requisitos valiados pela ferramenta, como é possível reparar na Figura 43, com apenas alguns requisitos ainda não sendo cumpridos por falta de algumas informações extras, como ícones, por exemplo, que permitem que os navegadores (em especial, navegadores projetados para

data-measurement/mobile-page-speed-new-industry-benchmarks/

[13] https://webpack.js.org/

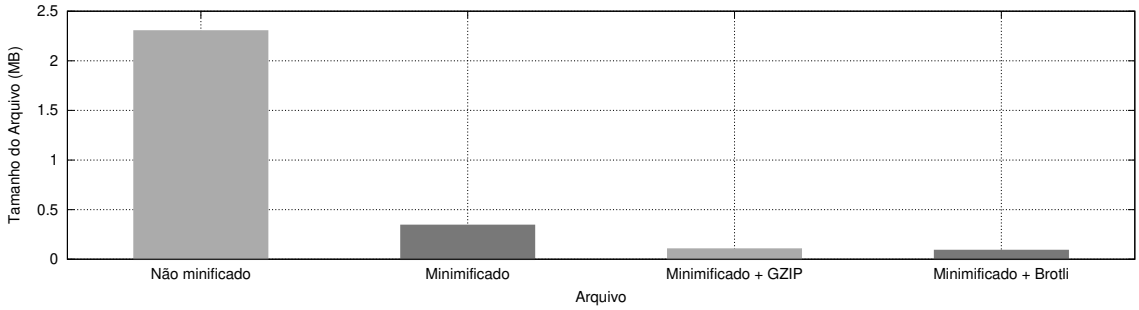

Figura 42. Comparação do tamanho do maior arquivo do *frontend* ao aplicar técnicas para diminuir a quantidade de dados transferida entre o servidor e o cliente

smartphones) tratem o *frontend* quase que como um aplicativo próprio.

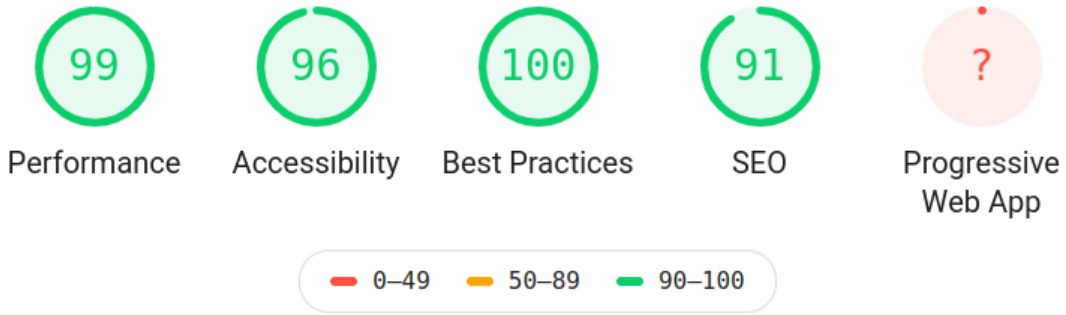

Figura 43. Resultado da análise da página de plano com a ferramenta *Lighthouse*.

Durante a execução da análise, que segundo a premissa do *Lighthouse* deve representar, de maneira aproximada, os aspectos de qualidade de um app da *Web*, foi possível observar que o sistema não avalia apenas aspectos próprios do *frontend*, mas também algumas configurações próprias do *backend*, como comunicação com o servidor através do protocolo HTTP 2.0 e também uso de conexão criptografada, por exemplo, que são necessários para que alguns recursos do próprio *frontend* sejam liberados pelo próprio navegador, afim de garantir a própria segurança do usuário.

Por fim, por não ter chegado à nota máxima em alguns dos aspectos avaliados, o *Lighthouse* acabou deixando um conjunto de sugestões para melhoria na aplicação:

- Melhorar contraste entre cor da fonte e cor de fundo no cabeçalho;
- Pré-carregar determinados recursos, como fontes, por exemplo, antecipadamente;
- Configurar determinadas páginas para funcionarem *offline*;
- Adicionar descrição das páginas para reconhecimento pelos motores de busca;
- Configurar período de *cache* para os recursos estáticos da página.

Além disso, boa parte dos testes que o *Lighthouse* possui não podem ser executados de forma automática, e portanto não foram avaliados para a realização desta análise.

7. Conclusão

Neste trabalho, foi apresentado a proposta e o protótipo de um simulador de matrícula multi-institucional, com suporte a lidar com problemas complexos como validação de pré-requisitos e armazenamento de grande quantidade de dados sobre cursos e ofertas de turmas. Para lidar com tudo isso, foi desenvolvido uma estrutura flexível, altamente paralela e capaz de lidar com os diferentes aspectos das universidades de modo genérico, mas garantindo o acesso total à informação coletada. Com a proposta apresentada, é possível que os alunos criem planos para o próximo período letivo facilmente, tanto no computador quanto em um *smartphone*, para qualquer uma das universidades cadastradas no sistema (inicialmente, UFSC e USP).

Além disso, também é permitido que usuários externos cadastrem novas universidades dentro do sistema, algo que nenhum dos outros sistemas avaliados suporta. Dessa forma, além de fornecer, de imediato, um serviço aos alunos das universidades já cadastradas, é possível expandir ainda mais o suporte a outras universidades, sem a necessidade de desenvolvimento de mais sistemas complexos e isolados apenas para permitir esta tarefa.

A partir dos testes realizados, foi possível verificar que o sistema cumpriu, também, os requisitos não-funcionais definidos na etapa de levantamento de requisitos, o que permite que o sistema execute em uma grande quantidade de configurações e também conte com diversas opções para otimizar o sistema para diferentes situações.

7.1. Trabalhos Futuros

Com o desenvolvimento do sistema, foi observado a existência de algumas possibilidades de trabalhos futuros que podem ser realizados visando a continuidade deste trabalho:

- Integração direta entre o sistema e sistemas acadêmicos de diferentes universidades através do uso de adaptadores para realização da matrícula através do simulador;
- Desenvolvimento de módulo capaz de computar matrículas válidas de acordo com as regras de uma universidade eliminando (ou ao menos diminuindo) a necessidade de desenvolver um sistema de matrícula próprio;
- Desenvolvimento de uma suite de testes sofisticada para garantir o funcionamento do sistema nas mais diferentes situações;
- Desenvolvimento de um sistema de notificações capaz de trabalhar na escala necessária, sem ter seu desempenho prejudicado.

Referências

[Alabbas and Bell 2017] Alabbas, A. and Bell, J. (2017). Indexed database API 2.0. W3C proposed reccommendation, W3C. https://www.w3.org/TR/2017/PR-IndexedDB-2-20171116/.

[Ater 2017] Ater, T. (2017). *Building Progressive Web Apps: Bringing the Power of Native to the Browser*. O'Reilly Media.

[Hickson et al. 2014] Hickson, I., Berjon, R., Navara, E. D., Faulkner, S., Pfeiffer, S., O'Connor, T., and Leithead, T. (2014). HTML5. W3C recommendation, W3C. http://www.w3.org/TR/2014/REC-html5-20141028/.

[Jansen 2015] Jansen, R. (2015). *Learning TypeScript*. Packt Publishing.

[Jr. et al. 2017] Jr., T. A., Rivoal, F., and Etemad, E. (2017). CSS snapshot 2017. W3C note, W3C. https://www.w3.org/TR/2017/NOTE-css-2017-20170131/.

[Mew 2015] Mew, K. (2015). *Learning Material Design*. Packt Publishing.

[Pike 2009] Pike, R. (2009). The go programming language. *Talk given at Google's Tech Talks*.

[Russell et al. 2017] Russell, A., Archibald, J., Song, J., and Kruisselbrink, M. (2017). Service workers 1. W3C working draft, W3C. https://www.w3.org/TR/2017/WD-service-workers-1-20171102/.

APÊNDICE B – Código Fonte

Atenção: Todo o código relacionado à este trabalho está disponível também no endereço <https://github.com/gurudamatricula>

B.1 Backend

Arquivo go.mod

```
1  module github.com/fjorgemota/gurudamatricula
2
3  require (
4          cloud.google.com/go v0.46.2
5          cloud.google.com/go/datastore v1.0.0
6          code.fjorgemota.com/fjorgemota/perm-go
           ↪  v0.0.0-20160616220446-f725dc12671a
7          github.com/99designs/gqlgen v0.9.3
8          github.com/AndreasBriese/bbloom
           ↪  v0.0.0-20190825152654-46b345b51c96 // indirect
9          github.com/DataDog/zstd v1.4.4 // indirect
10         github.com/NYTimes/gziphandler v1.1.1
11         github.com/RoaringBitmap/roaring v0.4.20
12         github.com/SherClockHolmes/webpush-go v1.0.1
13         github.com/agnivade/levenshtein v1.0.2 // indirect
14         github.com/boltdb/bolt v1.3.1
15         github.com/davecgh/go-spew v1.1.1
16         github.com/dgraph-io/badger v1.6.0
17         github.com/dgraph-io/badger/v2 v2.0.0
18         github.com/dgraph-io/ristretto v0.0.0-20191114170855-99d1bbbf28e6
           ↪  // indirect
19         github.com/dgryski/go-farm v0.0.0-20191112170834-c2139c5d712b //
           ↪  indirect
20         github.com/dsnet/compress v0.0.1
21         github.com/fsnotify/fsnotify v1.4.7
22         github.com/glycerine/go-unsnap-stream
           ↪  v0.0.0-20190901134440-81cf024a9e0a // indirect
```

```
23          github.com/go-telegram-bot-api/telegram-bot-api
            ↪ v4.6.4+incompatible // indirect
24          github.com/gobuffalo/envy v1.7.1 // indirect
25          github.com/gobuffalo/logger v1.0.1 // indirect
26          github.com/gobuffalo/packr/v2 v2.6.0
27          github.com/gofrs/uuid v3.2.0+incompatible
28          github.com/golang/protobuf v1.3.2
29          github.com/gopherjs/gopherjs v0.0.0-20181103185306-d547d1d9531e
            ↪ // indirect
30          github.com/gorilla/csrf v1.6.1
31          github.com/gorilla/securecookie v1.1.1
32          github.com/gorilla/sessions v1.2.0 // indirect
33          github.com/gorilla/websocket v1.4.1 // indirect
34          github.com/hashicorp/golang-lru v0.5.3
35          github.com/justinas/alice v0.0.0-20171023064455-03f45bd4b7da
36          github.com/magiconair/properties v1.8.1 // indirect
37          github.com/markbates/goth v1.56.0
38          github.com/onsi/ginkgo v1.8.0
39          github.com/onsi/gomega v1.5.0
40          github.com/pelletier/go-toml v1.4.0 // indirect
41          github.com/pkg/errors v0.8.1
42          github.com/pquerna/otp v1.2.0 // indirect
43          github.com/qri-io/jsonschema v0.1.1
44          github.com/rs/cors v1.7.0
45          github.com/sirupsen/logrus v1.4.2
46          github.com/smartystreets/assertions
            ↪ v0.0.0-20190215210624-980c5ac6f3ac // indirect
47          github.com/speps/go-hashids v2.0.0+incompatible
48          github.com/spf13/afero v1.2.2 // indirect
49          github.com/spf13/jwalterweatherman v1.1.0 // indirect
50          github.com/spf13/pflag v1.0.5 // indirect
51          github.com/spf13/viper v1.4.0
52          github.com/technoweenie/multipartstreamer v1.0.1 // indirect
53          github.com/ugorji/go/codec v1.1.7
54          github.com/ulikunitz/xz v0.5.6
55          github.com/vektah/gqlparser v1.1.2
56          github.com/volatiletech/authboss v2.3.0+incompatible
57          github.com/volatiletech/authboss-clientstate
            ↪ v0.0.0-20190912194043-b5b6e0f4355e
```

```
58          github.com/volatiletech/authboss-renderer
            ↪  v0.0.0-20181105062701-4b64de40529a
59          github.com/willf/bloom v2.0.3+incompatible
60          go.etcd.io/bbolt v1.3.3
61          go.opencensus.io v0.22.1 // indirect
62          golang.org/x/crypto v0.0.0-20190911031432-227b76d455e7
63          golang.org/x/exp v0.0.0-20190912063710-ac5d2bfcbfe0 // indirect
64          golang.org/x/net v0.0.0-20191119073136-fc4aabc6c914
65          golang.org/x/oauth2 v0.0.0-20190604053449-0f29369cfe45
66          golang.org/x/sys v0.0.0-20191120155948-bd437916bb0e // indirect
67          golang.org/x/text v0.3.2
68          golang.org/x/tools v0.0.0-20191030211004-889af361d29c // indirect
69          google.golang.org/api v0.10.0
70          google.golang.org/appengine v1.6.2
71          google.golang.org/genproto v0.0.0-20190911173649-1774047e7e51
72          google.golang.org/grpc v1.23.1 // indirect
73          gopkg.in/telegram-bot-api.v4 v4.6.4
74  )
75
76  go 1.13
```

Arquivo main.go

```
1   package main
2
3   import (
4           "context"
5           "fmt"
6           "log"
7           "sync"
8
9           "github.com/fsnotify/fsnotify"
10          "github.com/spf13/viper"
11
12          "github.com/fjorgemota/gurudamatricula/config"
13  )
14
15  func main() {
16          cfg := viper.New()
17          cfg.SetConfigName("config")
```

```
18          cfg.AddConfigPath("/etc/gurudamatricula/")
19          cfg.AddConfigPath("$HOME/.gurudamatricula")
20          cfg.AddConfigPath(".")
21          err := cfg.ReadInConfig()
22          if _, ok := err.(viper.ConfigFileNotFoundError); ok {
23                  err = nil
24          }
25          if err == nil {
26                  err = config.LoadConfig(cfg)
27          }
28          if err != nil {
29                  log.Fatal(err)
30          }
31          server := config.GetServer()
32          wg := &sync.WaitGroup{}
33          fmt.Println("Starting server at ", server.Addr)
34          cfg.OnConfigChange(func(_ fsnotify.Event) {
35                  fmt.Println("Restarting server")
36                  if err == nil {
37                          err = config.LoadConfig(cfg)
38                  }
39                  if err != nil {
40                          log.Fatal(err)
41                  }
42                  if err == nil {
43                          server.Shutdown(context.Background())
44                          server = config.GetServer()
45                          fmt.Println("Starting server at ", server.Addr)
46                          config.ListenAndServe()
47                          wg.Add(1)
48                          wg.Done()
49                  }
50          })
51          cfg.WatchConfig()
52          if err == nil {
53                  wg.Add(1)
54                  config.ListenAndServe()
55          }
56          if err == nil {
```

```
57                  wg.Wait()
58          }
59  }
```

B.1.1 Pasta auth

Arquivo base.go

```
1   package auth
2
3   import (
4           "context"
5           "encoding/gob"
6           "net/http"
7
8           "github.com/fjorgemota/gurudamatricula/util/kv"
9   )
10
11  type Handler interface {
12          GetStore(context.Context) (kv.Store, error)
13          GetContext(req *http.Request) context.Context
14  }
15
16  func init() {
17          gob.Register(sessionEntry{})
18          gob.Register(sessionKeyValue{})
19  }
```

Arquivo body_reader.go

```
1   package auth
2
3   import (
4           "net/http"
5
6           "github.com/volatiletech/authboss"
7           "github.com/volatiletech/authboss/defaults"
8   )
9
10  type HTTPBodyReader struct {
```

```
11          ReadJSON *defaults.HTTPBodyReader
12          ReadForm *defaults.HTTPBodyReader
13  }
14
15  func (hbr HTTPBodyReader) Read(page string, r *http.Request)
    ↪  (authboss.Validator, error) {
16          if r.Method == "GET" {
17                  return hbr.ReadForm.Read(page, r)
18          }
19          return hbr.ReadJSON.Read(page, r)
20  }
21
22  func NewHTTPBodyReader() authboss.BodyReader {
23          readJSON := defaults.NewHTTPBodyReader(true, false)
24          readForm := defaults.NewHTTPBodyReader(false, false)
25          readJSON.Whitelist["register"] =
            ↪  append(readJSON.Whitelist["register"], "name")
26          readForm.Whitelist["register"] =
            ↪  append(readForm.Whitelist["register"], "name")
27          return HTTPBodyReader{
28                  ReadJSON: readJSON,
29                  ReadForm: readForm,
30          }
31  }
```

Arquivo oauth2_details.go

```
1   package auth
2
3   import (
4           "context"
5
6           "github.com/fjorgemota/gurudamatricula/util/coder"
7           "golang.org/x/oauth2"
8   )
9
10  const (
11          idField    = "id"
12          nameField  = "name"
13          emailField = "email"
```

```
14 )
15
16 const (
17         googleInfoEndpoint   =
           ↪  "https://www.googleapis.com/userinfo/v2/me"
18         facebookInfoEndpoint =
           ↪  "https://graph.facebook.com/me?fields=id,name,email"
19 )
20
21 func GoogleUserDetails(ctx context.Context, cfg oauth2.Config, token
   ↪  *oauth2.Token) (map[string]string, error) {
22         client := cfg.Client(ctx, token)
23         resp, err := client.Get(googleInfoEndpoint)
24         var response struct {
25                 ID    string `json:"id"`
26                 Name  string `json:"name"`
27                 Email string `json:"email"`
28         }
29         if err == nil {
30                 cod := coder.NewJSONCoder()
31                 err = cod.DecodeFrom(resp.Body, &response)
32         }
33         if err == nil {
34                 err = resp.Body.Close()
35         }
36         var result map[string]string
37         if err == nil {
38                 result = map[string]string{
39                         idField:    response.ID,
40                         nameField:  response.Name,
41                         emailField: response.Email,
42                 }
43         }
44         return result, err
45 }
46
47 func FacebookUserDetails(ctx context.Context, cfg oauth2.Config, token
   ↪  *oauth2.Token) (map[string]string, error) {
48         client := cfg.Client(ctx, token)
```

```
49          resp, err := client.Get(facebookInfoEndpoint)
50          var response struct {
51                  ID    string `json:"id"`
52                  Email string `json:"email"`
53                  Name  string `json:"name"`
54          }
55          if err == nil {
56                  cod := coder.NewJSONCoder()
57                  err = cod.DecodeFrom(resp.Body, &response)
58          }
59          if err == nil {
60                  err = resp.Body.Close()
61          }
62          var result map[string]string
63          if err == nil {
64                  result = map[string]string{
65                          idField:    response.ID,
66                          nameField:  response.Name,
67                          emailField: response.Email,
68                  }
69          }
70
71          return result, err
72  }
```

Arquivo redirector.go

```
1   package auth
2
3   import (
4           "net/http"
5
6           "github.com/volatiletech/authboss"
7   )
8
9   // Redirector for http requests
10  type Redirector struct {
11          Renderer authboss.Renderer
12  }
13
```

```
14 // NewRedirector constructor
15 func NewRedirector(renderer authboss.Renderer) *Redirector {
16         return &Redirector{Renderer: renderer}
17 }
18
19 // Redirect the client elsewhere. If it's an API request it will simply
   ↪  render
20 // a JSON response with information that should help a client to decide
   ↪  what
21 // to do.
22 func (r *Redirector) Redirect(w http.ResponseWriter, req *http.Request,
   ↪  ro authboss.RedirectOptions) error {
23         var redirectFunction = r.redirectNonAPI
24         if req.Method != "GET" {
25                 redirectFunction = r.redirectAPI
26         }
27         return redirectFunction(w, req, ro)
28 }
29
30 func (r Redirector) redirectAPI(w http.ResponseWriter, req *http.Request,
   ↪  ro authboss.RedirectOptions) error {
31         path := ro.RedirectPath
32         redir := req.FormValue(authboss.FormValueRedirect)
33         if len(redir) != 0 && ro.FollowRedirParam {
34                 path = redir
35         }
36
37         var status = "success"
38         var message string
39         if len(ro.Success) != 0 {
40                 message = ro.Success
41         }
42         if len(ro.Failure) != 0 {
43                 status = "failure"
44                 message = ro.Failure
45         }
46
47         data := authboss.HTMLData{
48                 "location": path,
```

```
49          }
50
51          data["status"] = status
52          if len(message) != 0 {
53                  data["message"] = message
54          }
55
56          body, mime, err := r.Renderer.Render(req.Context(), "redirect",
            ↪  data)
57          if err != nil {
58                  return err
59          }
60
61          if len(body) != 0 {
62                  w.Header().Set("Content-Type", mime)
63          }
64
65          if ro.Code != 0 {
66                  if ro.Code == http.StatusTemporaryRedirect || ro.Code ==
                    ↪  http.StatusPermanentRedirect {
67                          w.WriteHeader(http.StatusOK)
68                  } else {
69                          w.WriteHeader(ro.Code)
70                  }
71          }
72          _, err = w.Write(body)
73          return err
74  }
75
76  func (r Redirector) redirectNonAPI(w http.ResponseWriter, req
    ↪  *http.Request, ro authboss.RedirectOptions) error {
77          path := ro.RedirectPath
78          redir := req.FormValue(authboss.FormValueRedirect)
79          if len(redir) != 0 && ro.FollowRedirParam {
80                  path = redir
81          }
82
83          if len(ro.Success) != 0 {
```

```
84                  authboss.PutSession(w, authboss.FlashSuccessKey,
                    ↪  ro.Success)
85          }
86          if len(ro.Failure) != 0 {
87                  authboss.PutSession(w, authboss.FlashErrorKey,
                    ↪  ro.Failure)
88          }
89
90          http.Redirect(w, req, path, http.StatusFound)
91          return nil
92  }
```

Arquivo session.go

```
1   package auth
2
3   import (
4           "context"
5           "crypto/sha1"
6           "encoding/hex"
7           "net"
8           "net/http"
9           "strings"
10          "time"
11
12          uuid "github.com/gofrs/uuid"
13          "github.com/gorilla/securecookie"
14
15          "github.com/fjorgemota/gurudamatricula/util/kv"
16
17          "github.com/fjorgemota/gurudamatricula/util/clock"
18
19          "github.com/volatiletech/authboss"
20  )
21
22  const tableSession = "sessions"
23
24  type sessionKeyValue struct {
25          Key   string `datastore:"key,noindex" json:"key"`
26          Value string `datastore:"value,noindex" json:"value"`
```

```
27 }
28
29 type sessionEntry struct {
30         ID      string            `datastore:"id,noindex" json:"id"`
31         Version string            `datastore:"version,noindex"
           ↪  json:"version"`
32         SavedAt time.Time         `datastore:"saved_at,noindex"
           ↪  json:"saved_at"`
33         Entries []sessionKeyValue `datastore:"entries,noindex"
           ↪  json:"entries"`
34 }
35
36 type SessionCookieOptions struct {
37         Name     string
38         Domain   string
39         Path     string
40         HTTPOnly bool
41         Secure   bool
42         SameSite http.SameSite
43 }
44
45 type SessionOptions struct {
46         ExpireSession time.Duration
47         Cookie        SessionCookieOptions
48         Clock         clock.Clock
49         SecureCookie  *securecookie.SecureCookie
50 }
51
52 type sessionState struct {
53         values      map[string]string
54         cookieValue string
55         ip          string
56         version     string
57         ctx         context.Context
58 }
59
60 func (ss sessionState) Get(key string) (string, bool) {
61         var value string
62         var ok bool
```

```
63          if ss.values != nil {
64                  value, ok = ss.values[key]
65          }
66          return value, ok
67  }
68
69  type sessionHandler struct {
70          options SessionOptions
71          handler Handler
72  }
73
74  func (ss sessionHandler) generateID(ip, id string) string {
75          hash := sha1.Sum([]byte(ip + "|" + id))
76          return hex.EncodeToString(hash[:])
77  }
78
79  func (ss sessionHandler) getCookieValue(req *http.Request) (string,
    ↪  error) {
80          var cookieValue string
81          cookie, err := req.Cookie(ss.options.Cookie.Name)
82          if err == nil {
83                  cookieValue = cookie.Value
84                  if ss.options.SecureCookie != nil {
85                          var result string
86                          err =
                            ↪  ss.options.SecureCookie.Decode(ss.options.Cookie.Name,
                            ↪  cookieValue, &result)
87                          if err == nil {
88                                  cookieValue = result
89                          }
90                  }
91          }
92          return cookieValue, err
93  }
94
95  func (ss sessionHandler) getIP(req *http.Request) (string, error) {
96          var err error
97          ip := req.Header.Get("X-Forwarded-For")
98          if len(ip) == 0 {
```

```
99                  ip = req.RemoteAddr
100         } else {
101                 ip = strings.SplitN(ip, ", ", 1)[0]
102         }
103         ip, _, err = net.SplitHostPort(ip)
104         return ip, err
105 }
106
107 func (ss sessionHandler) ReadState(req *http.Request)
    ↪  (authboss.ClientState, error) {
108         var session sessionEntry
109         var values map[string]string
110         var store kv.Store
111         var ip string
112         ctx := ss.handler.GetContext(req)
113         cookieValue, err := ss.getCookieValue(req)
114         if err == nil {
115                 ip, err = ss.getIP(req)
116         }
117         if err == nil {
118                 store, err = ss.handler.GetStore(ctx)
119         }
120         if err == nil {
121                 id := ss.generateID(ip, cookieValue)
122                 err = store.Get(tableSession, id, &session)
123         }
124         limit := ss.options.Clock.Now().Add(-ss.options.ExpireSession)
125         if err == nil && session.SavedAt.After(limit) {
126                 values = make(map[string]string, len(session.Entries))
127                 for _, entry := range session.Entries {
128                         values[entry.Key] = entry.Value
129                 }
130         }
131         if err == http.ErrNoCookie || kv.IsKeyNotFoundError(err) {
132                 cookieValue = ""
133                 err = nil
134         }
135         state := sessionState{
136                 cookieValue: cookieValue,
```

```
137                 values:      values,
138                 version:     session.Version,
139                 ctx:         ctx,
140                 ip:          ip,
141         }
142         return state, err
143 }
144
145 func (ss sessionHandler) WriteState(response http.ResponseWriter, state
    ↪  authboss.ClientState, events []authboss.ClientStateEvent) error {
146         entry := state.(sessionState)
147         store, err := ss.handler.GetStore(entry.ctx)
148         if err == nil && entry.cookieValue == "" {
149                 var id uuid.UUID
150                 id, err = uuid.NewV4()
151                 if err == nil {
152                         entry.cookieValue = id.String()
153                 }
154         }
155         cookieValue := entry.cookieValue
156         id := ss.generateID(entry.ip, cookieValue)
157         if err == nil && ss.options.SecureCookie != nil {
158                 cookieValue, err =
                    ↪  ss.options.SecureCookie.Encode(ss.options.Cookie.Name,
                    ↪  cookieValue)
159         }
160         if err == nil {
161                 if entry.values == nil {
162                         entry.values = make(map[string]string,
                            ↪  len(events))
163                 }
164                 for _, event := range events {
165                         switch event.Kind {
166                         case authboss.ClientStateEventPut:
167                                 entry.values[event.Key] = event.Value
168                         case authboss.ClientStateEventDel:
169                                 delete(entry.values, event.Key)
170                         case authboss.ClientStateEventDelAll:
171                                 var zero struct{}
```

```
172                             whitelistMap := make(map[string]struct{})
173                             for _, key := range
                                ↪  strings.Split(event.Key, ",") {
174                                     whitelistMap[key] = zero
175                             }
176                             for key := range entry.values {
177                                     if _, ok := whitelistMap[key];
                                        ↪  !ok {
178                                             delete(entry.values, key)
179                                     }
180                             }
181                     }
182             }
183             err = store.Transaction([]kv.Reference{{Table:
                ↪  tableSession, Key: id}}, func(db kv.LimitedStore)
                ↪  error {
184                     var dbEntry sessionEntry
185                     localErr := db.Get(tableSession, id, &dbEntry)
186                     if entry.version == "" &&
                        ↪  kv.IsKeyNotFoundError(localErr) {
187                             localErr = nil
188                             dbEntry = sessionEntry{
189                                     ID: id,
190                             }
191                     }
192                     if localErr == nil && dbEntry.Version ==
                        ↪  entry.version {
193                             dbEntry.Entries = dbEntry.Entries[:0]
194                             dbEntry.SavedAt = ss.options.Clock.Now()
195                             for key, value := range entry.values {
196                                     dbEntry.Entries =
                                        ↪  append(dbEntry.Entries,
                                        ↪  sessionKeyValue{
197                                             Key:   key,
198                                             Value: value,
199                                     })
200                             }
201                             var version uuid.UUID
202                             version, localErr = uuid.NewV4()
```

```
203                                 if localErr == nil {
204                                         dbEntry.Version =
                                            ↪ version.String()
205                                         localErr = db.Set(tableSession,
                                            ↪ id, &dbEntry)
206                                 }
207                         }
208                         return localErr
209                 })
210         }
211         if err == nil {
212                 http.SetCookie(response, &http.Cookie{
213                         Name:     ss.options.Cookie.Name,
214                         Value:    cookieValue,
215                         MaxAge:   int(ss.options.ExpireSession /
                            ↪ time.Second),
216                         Expires:
                            ↪ ss.options.Clock.Now().Add(ss.options.ExpireSession),
217                         HttpOnly: ss.options.Cookie.HTTPOnly,
218                         Secure:   ss.options.Cookie.Secure,
219                         Path:     ss.options.Cookie.Path,
220                         Domain:   ss.options.Cookie.Domain,
221                         SameSite: ss.options.Cookie.SameSite,
222                 })
223         }
224         return err
225 }
226
227 var (
228         assertSession                                =
            ↪ sessionHandler{}
229         _                   authboss.ClientStateReadWriter = assertSession
230         assertSessionState                           =
            ↪ sessionState{}
231         _                   authboss.ClientState     =
            ↪ assertSessionState
232 )
233
234 func getSessionDefaultOptions(options SessionOptions) SessionOptions {
```

```
235         if options.ExpireSession == 0 {
236                 options.ExpireSession = time.Hour * 24 * 30
237         }
238         if options.Cookie.Path == "" {
239                 options.Cookie.Path = "/"
240         }
241         if options.Cookie.Name == "" {
242                 options.Cookie.Name = "session-id"
243         }
244         if options.Clock == nil {
245                 options.Clock = clock.NewRealClock()
246         }
247         return options
248 }
249
250 func NewSessionHandler(handler Handler, options SessionOptions)
    ↪  authboss.ClientStateReadWriter {
251         options = getSessionDefaultOptions(options)
252         return sessionHandler{
253                 handler: handler,
254                 options: options,
255         }
256 }
```

Arquivo storer.go

```
1  package auth
2
3  import (
4          "context"
5          "errors"
6          "time"
7
8          "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
9
10         "github.com/fjorgemota/gurudamatricula/models/keygen"
11
12         "github.com/fjorgemota/gurudamatricula/models"
13         "github.com/fjorgemota/gurudamatricula/util/clock"
14         "github.com/fjorgemota/gurudamatricula/util/kv"
```

```
15          "github.com/volatiletech/authboss"
16  )
17
18  type StorerOptions struct {
19          ExpireSelectors time.Duration
20          Clock           clock.Clock
21  }
22
23  type StorerHandler struct {
24          handler Handler
25          options StorerOptions
26  }
27
28  func (st StorerHandler) userTransaction(ctx context.Context, refs
    ↪  []kv.Reference, fn func(db kv.LimitedStore) error) error {
29          store, err := st.handler.GetStore(ctx)
30          if err == nil {
31                  err = store.Transaction(refs, fn)
32          }
33          return err
34  }
35
36  func (st StorerHandler) Load(ctx context.Context, pid string)
    ↪  (authboss.User, error) {
37          var result *models.User
38          key, err := keygen.GenerateUserID(pid)
39          loader := dataloaders.GetUserLoader(ctx)
40          if err == nil {
41                  result, err = loader.Load(key)
42          }
43          if kv.IsKeyNotFoundError(err) {
44                  err = authboss.ErrUserNotFound
45          }
46          return result, err
47  }
48
49  func (st StorerHandler) Save(ctx context.Context, user authboss.User)
    ↪  error {
50          var toDelete []kv.Reference
```

```
51          var store kv.Store
52          var err error
53          entity := user.(*models.User)
54          pid := entity.GetPID()
55          key, err := keygen.GenerateUserID(pid)
56          refs := []kv.Reference{
57                  {Table: models.TableUser, Key: key},
58          }
59          if err == nil {
60                  store, err = st.handler.GetStore(ctx)
61          }
62          if selector := entity.GetConfirmSelector(); err == nil &&
            ↪  len(selector) > 0 {
63                  selector, err = keygen.GenerateUserSelector("confirm",
                    ↪  selector)
64                  if err == nil && len(selector) > 0 {
65                          refs = append(refs, kv.Reference{Table:
                            ↪  models.TableUserConfirmSelector, Key:
                            ↪  selector})
66                  }
67          }
68          if selector := entity.GetRecoverSelector(); err == nil &&
            ↪  len(selector) > 0 {
69                  selector, err = keygen.GenerateUserSelector("recover",
                    ↪  selector)
70                  if err == nil && len(selector) > 0 {
71                          refs = append(refs, kv.Reference{Table:
                            ↪  models.TableUserRecoverSelector, Key:
                            ↪  selector})
72                  }
73          }
74          if err == nil {
75                  err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪  error {
76                          var tmp models.User
77                          localErr := db.Get(models.TableUser, key, &tmp)
78                          if localErr == nil {
79                                  toDelete, localErr = st.save(db, key,
                                    ↪  &tmp, entity)
```

```
80                          } else if kv.IsKeyNotFoundError(localErr) {
81                                  localErr = authboss.ErrUserNotFound
82                          }
83                          return localErr
84                  })
85          }
86          if err == nil {
87                  err = st.deleteReferences(store, toDelete)
88          }
89          return err
90  }
91
92  func (st StorerHandler) New(ctx context.Context) authboss.User {
93          return &models.User{}
94  }
95
96  func (st StorerHandler) save(store kv.LimitedStore, key string, oldEntity,
    ↪  entity *models.User) ([]kv.Reference, error) {
97          var toDelete []kv.Reference
98          err := store.Set(models.TableUser, key, entity)
99          now := st.options.Clock.Now()
100         if err == nil && oldEntity != nil {
101                 if selector := oldEntity.GetConfirmSelector(); err == nil
                    ↪  && len(selector) > 0 {
102                         toDelete = append(toDelete, kv.Reference{
103                                 Table: models.TableUserConfirmSelector,
104                                 Key:   selector,
105                         })
106                 }
107                 if selector := oldEntity.GetRecoverSelector(); err == nil
                    ↪  && len(selector) > 0 {
108                         toDelete = append(toDelete, kv.Reference{
109                                 Table: models.TableUserRecoverSelector,
110                                 Key:   selector,
111                         })
112                 }
113         }
114         if selector := entity.GetConfirmSelector(); err == nil &&
            ↪  len(selector) > 0 {
```

```
115                 var id string
116                 id, err = keygen.GenerateUserSelector("confirm",
                    ↪  selector)
117                 if err == nil && len(id) > 0 {
118                         err = store.Set(models.TableUserConfirmSelector,
                            ↪  id, &models.UserSelector{
119                                 UserID:       key,
120                                 Selector:     selector,
121                                 RegisteredAt: now,
122                         })
123                 }
124         }
125         if selector := entity.GetRecoverSelector(); err == nil &&
            ↪  len(selector) > 0 {
126                 var id string
127                 id, err = keygen.GenerateUserSelector("recover",
                    ↪  selector)
128                 if err == nil && len(id) > 0 {
129                         err = store.Set(models.TableUserRecoverSelector,
                            ↪  id, &models.UserSelector{
130                                 UserID:       key,
131                                 Selector:     selector,
132                                 RegisteredAt: now,
133                         })
134                 }
135         }
136         return toDelete, err
137 }
138
139 func (st StorerHandler) deleteReferences(store kv.LimitedStore, toDelete
    ↪  []kv.Reference) error {
140         var err error
141         for _, ref := range toDelete {
142                 if err == nil {
143                         err = store.Del(ref.Table, ref.Key)
144                 }
145         }
146         return err
147 }
```

```
148
149 func (st StorerHandler) Create(ctx context.Context, user authboss.User)
    ↪  error {
150         var err error
151         entity := user.(*models.User)
152         provider := entity.GetOAuth2Provider()
153         if len(provider) == 0 {
154                 provider = "local"
155         }
156         pid := entity.GetPID()
157         key, err := keygen.GenerateUserID(pid)
158         refs := []kv.Reference{
159                 {Table: models.TableUser, Key: key},
160         }
161         if selector := entity.GetConfirmSelector(); err == nil &&
            ↪  len(selector) > 0 {
162                 selector, err = keygen.GenerateUserSelector("confirm",
                    ↪  selector)
163                 if len(selector) > 0 && err == nil {
164                         refs = append(refs, kv.Reference{Table:
                            ↪  models.TableUserConfirmSelector, Key:
                            ↪  selector})
165                 }
166         }
167         if selector := entity.GetRecoverSelector(); err == nil &&
            ↪  len(selector) > 0 {
168                 selector, err = keygen.GenerateUserSelector("recover",
                    ↪  selector)
169                 if len(selector) > 0 && err == nil {
170                         refs = append(refs, kv.Reference{Table:
                            ↪  models.TableUserRecoverSelector, Key:
                            ↪  selector})
171                 }
172         }
173         if err == nil {
174                 err = st.userTransaction(ctx, refs, func(db
                    ↪  kv.LimitedStore) error {
175                         var tmp models.User
176                         localErr := db.Get(models.TableUser, key, &tmp)
```

```
177                         if localErr == nil {
178                                 localErr = authboss.ErrUserFound
179                         } else if kv.IsKeyNotFoundError(localErr) {
180                                 _, localErr = st.save(db, key, nil,
                                    ↪  entity)
181                         }
182                         return localErr
183                 })
184         }
185         return err
186 }
187
188 func (st StorerHandler) loadBySelector(ctx context.Context, selectorType,
    ↪  table, selectorValue string) (*models.User, error) {
189         var result *models.User
190         var userSelector models.UserSelector
191         var store kv.Store
192         id, err := keygen.GenerateUserSelector(selectorType,
            ↪  selectorValue)
193         if err == nil {
194                 store, err = st.handler.GetStore(ctx)
195         }
196         if err == nil {
197                 err = store.Get(table, id, &userSelector)
198         }
199         if err == nil {
200                 if userSelector.Selector == selectorValue &&
                    ↪  userSelector.UserID != "" {
201                         loader := dataloaders.GetUserLoader(ctx)
202                         result, err = loader.Load(userSelector.UserID)
203                 } else {
204                         err = errors.New("Invalid Selector")
205                 }
206         }
207         if kv.IsKeyNotFoundError(err) {
208                 err = authboss.ErrUserNotFound
209         }
210         if err != nil {
211                 result = nil
```

```
212         }
213         return result, err
214 }
215
216 func (st StorerHandler) LoadByConfirmSelector(ctx context.Context,
    ↪  selector string) (authboss.ConfirmableUser, error) {
217         user, err := st.loadBySelector(ctx, "confirm",
            ↪  models.TableUserConfirmSelector, selector)
218         if err == nil && user != nil && user.GetConfirmSelector() !=
            ↪  selector {
219                 user = nil
220                 err = authboss.ErrUserNotFound
221         }
222         return user, err
223 }
224
225 func (st StorerHandler) LoadByRecoverSelector(ctx context.Context,
    ↪  selector string) (authboss.RecoverableUser, error) {
226         user, err := st.loadBySelector(ctx, "recover",
            ↪  models.TableUserRecoverSelector, selector)
227         if err == nil && user != nil && user.GetRecoverSelector() !=
            ↪  selector {
228                 user = nil
229                 err = authboss.ErrUserNotFound
230         }
231         return user, err
232 }
233
234 func (st StorerHandler) AddRememberToken(ctx context.Context, pid, token
    ↪  string) error {
235         key, err := keygen.GenerateUserID(pid)
236         if err == nil {
237                 refs := []kv.Reference{{Table: models.TableUser, Key:
                    ↪  key}}
238                 err = st.userTransaction(ctx, refs, func(db
                    ↪  kv.LimitedStore) error {
239                         var user models.User
240                         localErr := db.Get(models.TableUser, key, &user)
241                         if localErr == nil {
```

```
242                                 user.RememberTokens =
                                    ↪ append(user.RememberTokens, token)
243                                 localErr = db.Set(models.TableUser, key,
                                    ↪ &user)
244                         } else if kv.IsKeyNotFoundError(localErr) {
245                                 localErr = authboss.ErrUserNotFound
246                         }
247                         return localErr
248                 })
249         }
250         return err
251 }
252
253 func (st StorerHandler) DelRememberTokens(ctx context.Context, pid
    ↪ string) error {
254         key, err := keygen.GenerateUserID(pid)
255         if err == nil {
256                 refs := []kv.Reference{{Table: models.TableUser, Key:
                    ↪ key}}
257                 err = st.userTransaction(ctx, refs, func(db
                    ↪ kv.LimitedStore) error {
258                         var user models.User
259                         localErr := db.Get(models.TableUser, key, &user)
260                         if localErr == nil {
261                                 user.RememberTokens =
                                    ↪ user.RememberTokens[:0]
262                                 localErr = db.Set(models.TableUser, key,
                                    ↪ &user)
263                         } else if kv.IsKeyNotFoundError(localErr) {
264                                 localErr = authboss.ErrUserNotFound
265                         }
266                         return localErr
267                 })
268         }
269         return err
270 }
271
272 func (st StorerHandler) UseRememberToken(ctx context.Context, pid, token
    ↪ string) error {
```

```
273         key, err := keygen.GenerateUserID(pid)
274         if err == nil {
275                 refs := []kv.Reference{{Table: models.TableUser, Key:
                    ↪ key}}
276                 err = st.userTransaction(ctx, refs, func(db
                    ↪ kv.LimitedStore) error {
277                         var user models.User
278                         localErr := db.Get(models.TableUser, key, &user)
279                         if localErr == nil {
280                                 tokens := user.RememberTokens[:0]
281                                 for _, rememberToken := range
                                    ↪ user.RememberTokens {
282                                         if rememberToken != token {
283                                                 tokens = append(tokens,
                                                    ↪ rememberToken)
284                                         }
285                                 }
286                                 if len(tokens) < len(user.RememberTokens)
                                    ↪ {
287                                         user.RememberTokens = tokens
288                                         localErr =
                                            ↪ db.Set(models.TableUser, key,
                                            ↪ &user)
289                                 } else {
290                                         localErr =
                                            ↪ authboss.ErrTokenNotFound
291                                 }
292                         } else if kv.IsKeyNotFoundError(localErr) {
293                                 localErr = authboss.ErrUserNotFound
294                         }
295                         return localErr
296                 })
297         }
298         return err
299 }
300
301 func (st StorerHandler) NewFromOAuth2(ctx context.Context, provider
    ↪ string, details map[string]string) (authboss.OAuth2User, error) {
302         var user *models.User
```

```
303         var err error
304         if provider != "facebook" && provider != "google" {
305                 err = errors.New("unknown provider")
306         }
307         id := details[idField]
308         pid := authboss.MakeOAuth2PID(provider, id)
309         var key string
310         if err == nil {
311                 key, err = keygen.GenerateUserID(pid)
312         }
313         refs := []kv.Reference{
314                 {Table: models.TableUser, Key: key},
315         }
316         if err == nil {
317                 err = st.userTransaction(ctx, refs, func(db
                ↪  kv.LimitedStore) error {
318                         user = &models.User{}
319                         localErr := db.Get(models.TableUser, key, user)
320                         if kv.IsKeyNotFoundError(localErr) {
321                                 user = &models.User{}
322                                 localErr = nil
323                         }
324                         return localErr
325                 })
326         }
327         if err == nil && user != nil {
328                 user.ID = pid
329                 user.OAuth2UID = id
330                 user.Name = details[nameField]
331                 user.Email = details[emailField]
332                 user.Confirmed = true
333         }
334         return user, err
335 }
336
337 func (st StorerHandler) SaveOAuth2(ctx context.Context, user
    ↪  authboss.OAuth2User) error {
338         var store kv.Store
```

```
339         pid := authboss.MakeOAuth2PID(user.GetOAuth2Provider(),
            ↪ user.GetOAuth2UID())
340         key, err := keygen.GenerateUserID(pid)
341         if err == nil {
342                 store, err = st.handler.GetStore(ctx)
343         }
344         if err == nil {
345                 err = store.Set(models.TableUser, key, user)
346         }
347         return err
348 }
349
350 var (
351         assertStorer = StorerHandler{}
352
353         _ authboss.CreatingServerStorer    = assertStorer
354         _ authboss.ConfirmingServerStorer  = assertStorer
355         _ authboss.RecoveringServerStorer  = assertStorer
356         _ authboss.RememberingServerStorer = assertStorer
357         _ authboss.OAuth2ServerStorer      = assertStorer
358 )
359
360 func getStorerDefaultOptions(options StorerOptions) StorerOptions {
361         if options.ExpireSelectors == 0 {
362                 options.ExpireSelectors = time.Hour * 24
363         }
364         if options.Clock == nil {
365                 options.Clock = clock.NewRealClock()
366         }
367         return options
368 }
369
370 func NewStorerHandler(handler Handler, options StorerOptions)
    ↪ StorerHandler {
371         options = getStorerDefaultOptions(options)
372         return StorerHandler{
373                 handler: handler,
374                 options: options,
375         }
```

```
376 }
```

B.1.2 Pasta config

Arquivo auth.go

```
1  package config
2
3  import (
4          "io"
5          "net/smtp"
6          "time"
7
8          "github.com/fjorgemota/gurudamatricula/auth"
9          "github.com/gorilla/securecookie"
10         "github.com/markbates/goth/providers/google"
11         "github.com/spf13/viper"
12         "github.com/volatiletech/authboss"
13         abclientstate "github.com/volatiletech/authboss-clientstate"
14         abrenderer "github.com/volatiletech/authboss-renderer"
15         _ "github.com/volatiletech/authboss/auth"
16         _ "github.com/volatiletech/authboss/confirm"
17         "github.com/volatiletech/authboss/defaults"
18         _ "github.com/volatiletech/authboss/logout"
19         _ "github.com/volatiletech/authboss/oauth2"
20         _ "github.com/volatiletech/authboss/recover"
21         _ "github.com/volatiletech/authboss/register"
22         _ "github.com/volatiletech/authboss/remember"
23         "golang.org/x/oauth2"
24         "golang.org/x/oauth2/facebook"
25 )
26
27 var authInstance *authboss.Authboss
28
29 func getSMTPAuth(cfg *viper.Viper) smtp.Auth {
30         var result smtp.Auth
31         switch cfg.GetString("auth.mailer.auth.type") {
32         default:
33                 fallthrough
34         case "plain":
```

```
35                  identity := cfg.GetString("auth.mailer.auth.identity")
36                  username := cfg.GetString("auth.mailer.auth.username")
37                  password := cfg.GetString("auth.mailer.auth.password")
38                  host := cfg.GetString("auth.mailer.auth.host")
39                  result = smtp.PlainAuth(identity, username, password,
                    ↪  host)
40          case "cram-md5":
41                  username := cfg.GetString("auth.mailer.auth.username")
42                  password := cfg.GetString("auth.mailer.auth.password")
43                  result = smtp.CRAMMD5Auth(username, password)
44          }
45          return result
46  }
47
48  func getAuthBossMailer(cfg *viper.Viper) (authboss.Mailer, error) {
49          var result authboss.Mailer
50          var err error
51          switch cfg.GetString("auth.mailer.type") {
52          case "smtp":
53                  server := cfg.GetString("auth.mailer.server")
54                  result = defaults.NewSMTPMailer(server, getSMTPAuth(cfg))
55          case "log":
56                  var output io.Writer
57                  output, err =
                    ↪  getLoggerOutput(cfg.GetString("auth.mailer.output"))
58                  if err == nil {
59                          result = defaults.NewLogMailer(output)
60                  }
61          }
62          return result, err
63  }
64
65  func getAuthBossProviders(cfg *viper.Viper)
    ↪  map[string]authboss.OAuth2Provider {
66          providers := make(map[string]authboss.OAuth2Provider)
67          if fbCfg := cfg.Sub("auth.oauth2.facebook"); fbCfg != nil {
68                  providers["facebook"] = authboss.OAuth2Provider{
69                          FindUserDetails: auth.FacebookUserDetails,
70                          OAuth2Config: &oauth2.Config{
```

```
71                                      ClientID:
                                        ↪  fbCfg.GetString("client_id"),
72                                      ClientSecret:
                                        ↪  fbCfg.GetString("client_secret"),
73                                      Endpoint:     facebook.Endpoint,
74                                      Scopes:       []string{"email"},
75                              },
76                      }
77              }
78              if googleCfg := cfg.Sub("auth.oauth2.google"); googleCfg != nil {
79                      providers["google"] = authboss.OAuth2Provider{
80                              FindUserDetails: auth.GoogleUserDetails,
81                              OAuth2Config: &oauth2.Config{
82                                      ClientID:
                                        ↪  googleCfg.GetString("client_id"),
83                                      ClientSecret:
                                        ↪  googleCfg.GetString("client_secret"),
84                                      Endpoint:     google.Endpoint,
85                                      Scopes:       []string{"email",
                                        ↪  "https://www.googleapis.com/auth/userinfo.profile"},
86                              },
87                      }
88              }
89              return providers
90      }
91      func getAuthBossInstance(cfg *viper.Viper) (*authboss.Authboss, error) {
92              var err error
93              var instance *authboss.Authboss
94              var dependencies handlerDependencies
95              dependencies.getStore, err =
                ↪  getStoreCallback(cfg.Sub("auth.store"))
96              if err == nil {
97                      instance = authboss.New()
98                      clk := GetClock()
99                      var blockKey []byte
100                     if cfg.IsSet("auth.secure_cookie.block_key") {
101                             blockKey =
                                ↪  []byte(cfg.GetString("auth.secure_cookie.block_key"))
102                     }
```

```
103                 cookie :=
                    ↪ securecookie.New([]byte(cfg.GetString("auth.secure_cookie.hash_
                    ↪ blockKey)
104                 instance.Config.Storage.Server =
                    ↪ auth.NewStorerHandler(dependencies,
                    ↪ auth.StorerOptions{
105                         Clock:           clk,
106                         ExpireSelectors:
                            ↪ cfg.GetDuration("auth.storer.expire_selectors"),
107                 })
108                 instance.Config.Storage.SessionState =
                    ↪ auth.NewSessionHandler(dependencies,
                    ↪ auth.SessionOptions{
109                         Clock:         clk,
110                         ExpireSession:
                            ↪ cfg.GetDuration("auth.session.expire_session"),
111                         SecureCookie:  cookie,
112                         Cookie: auth.SessionCookieOptions{
113                                 Path:
                                    ↪ cfg.GetString("auth.session.cookie.path"),
114                                 HTTPOnly:
                                    ↪ cfg.GetBool("auth.session.cookie.http_only"),
115                                 Secure:
                                    ↪ cfg.GetBool("auth.session.cookie.secure"),
116                         },
117                 })
118                 instance.Config.Modules.OAuth2Providers =
                    ↪ getAuthBossProviders(cfg)
119                 instance.Config.Modules.LogoutMethod = "POST"
120                 instance.Config.Mail.From =
                    ↪ cfg.GetString("auth.mailer.from.email")
121                 instance.Config.Mail.FromName =
                    ↪ cfg.GetString("auth.mailer.from.name")
122                 instance.Config.Mail.RootURL =
                    ↪ cfg.GetString("auth.mailer.root_url")
123                 instance.Config.Mail.SubjectPrefix =
                    ↪ cfg.GetString("auth.mailer.subject_prefix")
124                 instance.Config.Storage.CookieState =
                    ↪ abclientstate.NewCookieStorerFromExisting(cookie)
```

```
125                 instance.Config.Paths.Mount = GetRoute("auth")
126                 instance.Config.Paths.RootURL = GetRootURL()
127                 instance.Config.Core.Mailer, err = getAuthBossMailer(cfg)
128                 instance.Config.Core.ViewRenderer =
                    ↪  defaults.JSONRenderer{}
129                 instance.Config.Core.MailRenderer =
                    ↪  abrenderer.NewEmail(instance.Config.Paths.Mount,
                    ↪  cfg.GetString("auth.mailer.template_dir"))
130                 instance.Config.Core.Router = defaults.NewRouter()
131                 if err == nil {
132                         var output io.Writer
133                         output, err =
                            ↪  getLoggerOutput(cfg.GetString("auth.log.output"))
134                         if err == nil {
135                                 logger := defaults.NewLogger(output)
136                                 instance.Config.Core.ErrorHandler =
                                    ↪  defaults.NewErrorHandler(logger)
137                                 instance.Config.Core.Logger = logger
138                         }
139                 }
140                 instance.Config.Core.Responder =
                    ↪  defaults.NewResponder(instance.Config.Core.ViewRenderer)
141                 instance.Config.Core.Redirector =
                    ↪  auth.NewRedirector(instance.Config.Core.ViewRenderer)
142                 instance.Config.Core.BodyReader =
                    ↪  auth.NewHTTPBodyReader()
143                 if err == nil {
144                         err = instance.Init()
145                 }
146         }
147         return instance, err
148 }
149
150 func initAuth(cfg *viper.Viper) error {
151         var err error
152         cfg.SetDefault("auth.session.expire_session", 7*24*time.Hour)
153         cfg.SetDefault("auth.storer.expire_selectors", 24*time.Hour)
154         cfg.SetDefault("auth.session.cookie.path", "/")
155         cfg.SetDefault("auth.session.cookie.http_only", true)
```

```
156         cfg.SetDefault("auth.session.cookie.secure", GetRootURLScheme()
            ↪  == "https")
157         cfg.SetDefault("auth.mailer.type", "log")
158         cfg.SetDefault("auth.mailer.output", "stderr")
159         cfg.SetDefault("auth.log.output", "stderr")
160         authInstance, err = getAuthBossInstance(cfg)
161         return err
162 }
163
164 func GetAuthBoss() *authboss.Authboss {
165         return authInstance
166 }
```

Arquivo base.go

```
1  // +build !appengine
2
3  package config
4
5  import (
6          "github.com/spf13/viper"
7  )
8
9  func initBasic(cfg *viper.Viper) error {
10         err := initRoutes(cfg)
11         if err == nil {
12                 err = initCors(cfg)
13         }
14         if err == nil {
15                 err = initCsrf(cfg)
16         }
17         if err == nil {
18                 err = initGzip(cfg)
19         }
20         if err == nil {
21                 err = initCache(cfg)
22         }
23         if err == nil {
24                 err = initCoder(cfg)
25         }
```

```
26          if err == nil {
27                  err = initLogger(cfg)
28          }
29          if err == nil {
30                  err = initHTTPClient(cfg)
31          }
32          if err == nil {
33                  err = initManager(cfg)
34          }
35          if err == nil {
36                  err = initStore(cfg)
37          }
38          if err == nil {
39                  err = initTaskQueue(cfg)
40          }
41          if err == nil {
42                  err = initPipeline(cfg)
43          }
44          if err == nil {
45                  err = initIndex(cfg)
46          }
47          if err == nil {
48                  err = initAuth(cfg)
49          }
50          return err
51  }
52
53  func initApp(cfg *viper.Viper) error {
54          err := initDDifferHandler(cfg)
55          if err == nil {
56                  err = initRedirectorController(cfg)
57          }
58          if err == nil {
59                  err = initImporterHandler(cfg)
60          }
61          if err == nil {
62                  err = initFetcher(cfg)
63          }
64          if err == nil {
```

```
65                  err = initUFSC2OffersBot(cfg)
66          }
67          if err == nil {
68                  err = initUFSC2HabilitationsBot(cfg)
69          }
70          if err == nil {
71                  err = initUSP2OffersBot(cfg)
72          }
73          if err == nil {
74                  err = initUSP2HabilitationsBot(cfg)
75          }
76          if err == nil {
77                  err = initDataloaderMiddleware(cfg)
78          }
79          if err == nil {
80                  err = initGraphQLController(cfg)
81          }
82          if err == nil {
83                  err = initStaticFileSystem(cfg)
84          }
85          if err == nil {
86                  err = initRouter(cfg)
87          }
88          if err == nil {
89                  err = initServer(cfg)
90          }
91          return err
92  }
93
94  func LoadConfig(cfg *viper.Viper) error {
95          closeAll()
96          err := initBasic(cfg)
97          if err == nil {
98                  err = initApp(cfg)
99          }
100         return err
101 }
102
103 func InitDefaultConfig() error {
```

```
104         return LoadConfig(viper.New())
105 }
106
107 func sub(cfg *viper.Viper, sub string) *viper.Viper {
108         result := cfg.Sub(sub)
109         if result == nil {
110                 result = viper.New()
111         }
112         return result
113 }
```

Arquivo bot.go

```
1  package config
2
3  import (
4          "errors"
5          "net/http"
6
7          "github.com/fjorgemota/gurudamatricula/controllers"
8          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
9          "github.com/fjorgemota/gurudamatricula/robot/publisher"
10         "github.com/fjorgemota/gurudamatricula/util/pipeline"
11         "github.com/spf13/viper"
12 )
13
14 func getBotDependencies(cfg *viper.Viper) (handlerDependencies, error) {
15         cfg = cfg.Sub("dependencies")
16         if cfg == nil {
17                 cfg = viper.New()
18         }
19         var dependencies handlerDependencies
20         var err error
21         dependencies.getHTTPClient =
           ↪  getHTTPClientCallback(cfg.Sub("client"))
22         dependencies.getStore, err = getStoreCallback(cfg.Sub("store"))
23         if err == nil {
24                 dependencies.streaming, err =
                   ↪  getStreamingCoderInstance(cfg.Sub("streaming_coder"))
25         }
```

```
26          if err == nil {
27                  dependencies.getManager, err =
                    ↪ getManagerCallback(cfg.Sub("manager"))
28          }
29          if err == nil {
30                  dependencies.getLogger, err =
                    ↪ getLoggerCallback(cfg.Sub("log"))
31          }
32          return dependencies, err
33  }
34
35  func hasRequiredPublisherOptions(cfg *viper.Viper, requiredOptions
    ↪ []string) bool {
36          if cfg != nil {
37                  cfg = cfg.Sub("publisher")
38          }
39          if cfg == nil {
40                  cfg = viper.New()
41          }
42          result := true
43          if len(requiredOptions) == 0 {
44                  requiredOptions = []string{"notification_url",
                    ↪ "university_id", "secret", "start_secret",
                    ↪ "notification_url", "delete_secret"}
45          }
46          for _, option := range requiredOptions {
47                  result = result && cfg.IsSet(option)
48          }
49          return result
50  }
51
52  func getPublisherOptions(cfg *viper.Viper, source ddiffer.SourceType)
    ↪ publisher.Options {
53          cfg = cfg.Sub("publisher")
54          if cfg == nil {
55                  cfg = viper.New()
56          }
57          var options publisher.Options
58          options.SourceType = source
```

```
59          options.NotificationURL = cfg.GetString("notification_url")
60          options.UniversityID = cfg.GetString("university_id")
61          options.Secret = cfg.GetString("secret")
62          options.Tasks.DeleteFile = cfg.GetString("tasks.delete_file")
63          options.Tasks.Notify = cfg.GetString("tasks.notify")
64          options.Tasks.Start = cfg.GetString("tasks.start")
65          return options
66  }
67
68  func getPublisherController(cfg *viper.Viper, sourceType
    ↪  ddiffer.SourceType, dispatcher pipeline.Dispatcher, bot publisher.Bot,
    ↪  dependencies handlerDependencies) (http.Handler, error) {
69          var result http.Handler
70          publisherOptions := getPublisherOptions(cfg, sourceType)
71          instance, err := publisher.NewBotPublisher(dispatcher, bot,
            ↪  dependencies, publisherOptions)
72          secrets := controllers.PublisherSecrets{
73                  StartSecret:  cfg.GetString("publisher.start_secret"),
74                  DeleteSecret: cfg.GetString("publisher.delete_secret"),
75          }
76          if err == nil && bot != nil && len(secrets.StartSecret) == 0 {
77                  err = errors.New("Start secret cannot be empty")
78          }
79          if err == nil && len(secrets.DeleteSecret) == 0 {
80                  err = errors.New("Delete secret cannot be empty")
81          }
82          if err == nil && secrets.StartSecret == secrets.DeleteSecret {
83                  err = errors.New("Start and Delete secrets cannot be
                    ↪  equal")
84          }
85          if err == nil {
86                  result = controllers.Publisher(dependencies, instance,
                    ↪  secrets)
87          }
88          return result, err
89  }
```

Arquivo cache.go

```
1  package config
```

```
2
3  import (
4          "time"
5
6          "github.com/fjorgemota/gurudamatricula/util/cache"
7          "github.com/fjorgemota/gurudamatricula/util/coder"
8          "github.com/fjorgemota/gurudamatricula/util/file"
9          "github.com/spf13/viper"
10         "golang.org/x/net/context"
11 )
12
13 var getCache func(ctx context.Context) (cache.Cache, error)
14
15 func getCacheCallback(cfg *viper.Viper) (func(ctx context.Context)
   ↪  (cache.Cache, error), error) {
16         if cfg == nil {
17                 return getCache, nil
18         }
19         var instance cache.Cache
20         var err error
21         switch cfg.GetString("type") {
22         case "lru":
23                 cfg.SetDefault("capacity", 10000)
24                 var cod coder.Coder
25                 cod, err = getCoderInstance(cfg.Sub("coder"))
26                 if err == nil {
27                         instance, err = cache.NewLRUCache(GetClock(), cod,
                           ↪  cfg.GetInt("capacity"),
                           ↪  cfg.GetDuration("default_expiration"))
28                 }
29         case "nil":
30                 instance = cache.NewNilCache()
31         case "file":
32                 var getManager func(ctx context.Context) (file.Manager,
                   ↪  error)
33                 var cod coder.Coder
34                 getManager, err = getManagerCallback(cfg.Sub("manager"))
35                 if err == nil {
36                         cod, err = getCoderInstance(cfg.Sub("coder"))
```

```
37                }
38                defaultExpiration :=
                  ↪  cfg.GetDuration("default_expiration")
39                clk := GetClock()
40                return func(ctx context.Context) (cache.Cache, error) {
41                        var result cache.Cache
42                        manager, localErr := getManager(ctx)
43                        if localErr == nil {
44                                result = cache.NewFileCache(manager, cod,
                                  ↪  defaultExpiration, clk)
45                        }
46                        return result, localErr
47                }, err
48        case "appengine":
49                var cod coder.Coder
50                cod, err = getCoderInstance(cfg.Sub("coder"))
51                duration := cfg.GetDuration("default_expiration")
52                return func(ctx context.Context) (cache.Cache, error) {
53                        return cache.NewAppEngineCache(ctx, cod,
                          ↪  duration), err
54                }, err
55        case "multi":
56                caches := cfg.Sub("caches")
57                order := cfg.GetStringSlice("order")
58                var functions []func(ctx context.Context) (cache.Cache,
                  ↪  error)
59                for _, name := range order {
60                        var fn func(ctx context.Context) (cache.Cache,
                          ↪  error)
61                        if err == nil {
62                                fn, err =
                                  ↪  getCacheCallback(caches.Sub(name))
63                        }
64                        if err == nil {
65                                functions = append(functions, fn)
66                        }
67                }
68                return func(ctx context.Context) (cache.Cache, error) {
```

```
69                          instances := make([]cache.Cache, 0,
                            ↪  len(functions))
70                          var result cache.Cache
71                          var localErr error
72                          for _, fn := range functions {
73                                  var cac cache.Cache
74                                  if localErr == nil {
75                                          cac, localErr = fn(ctx)
76                                  }
77                                  if localErr == nil {
78                                          instances = append(instances,
                                            ↪  cac)
79                                  }
80                          }
81                          if localErr == nil {
82                                  result =
                                    ↪  cache.NewMultiCache(instances...)
83                          }
84                          return result, localErr
85                  }, err
86          }
87          return func(ctx context.Context) (cache.Cache, error) {
88                  return instance, err
89          }, err
90  }
91
92  func initCache(cfg *viper.Viper) error {
93          cfg.SetDefault("cache.type", "nil")
94          cfg.SetDefault("cache.default_expiration", 24*time.Hour)
95          var err error
96          getCache, err = getCacheCallback(cfg.Sub("cache"))
97          return err
98  }
99
100 func GetCache(ctx context.Context) (cache.Cache, error) {
101         return getCache(ctx)
102 }
```

Arquivo client.go

```
1  package config
2
3  import (
4          "net"
5          "net/http"
6          "time"
7
8          "github.com/spf13/viper"
9          "golang.org/x/net/context"
10         "google.golang.org/appengine/urlfetch"
11 )
12
13 var getHTTPClient func(ctx context.Context, jar http.CookieJar)
   ↪  *http.Client
14
15 func getHTTPClientCallback(cfg *viper.Viper) func(ctx context.Context,
   ↪  jar http.CookieJar) *http.Client {
16         if cfg == nil {
17                 cfg = viper.New()
18         }
19         var result func(ctx context.Context, jar http.CookieJar)
           ↪  *http.Client
20         timeout := cfg.GetDuration("duration")
21         switch cfg.GetString("platform") {
22         case "appengine":
23                 result = func(ctx context.Context, jar http.CookieJar)
                   ↪  *http.Client {
24                         client := urlfetch.Client(ctx)
25                         client.Timeout = timeout
26                         client.Jar = jar
27                         return client
28                 }
29         default:
30                 cfg.SetDefault("max_idle_connections", 100)
31                 transport := &http.Transport{
32                         Proxy: http.ProxyFromEnvironment,
33                         DialContext: (&net.Dialer{
34                                 Timeout:   30 * time.Second,
35                                 KeepAlive: 30 * time.Second,
```

```
36                              DualStack: true,
37                      }).DialContext,
38                      MaxIdleConns:
                        ↪  cfg.GetInt("max_idle_connections"),
39                      IdleConnTimeout:       90 * time.Second,
40                      TLSHandshakeTimeout:   10 * time.Second,
41                      ExpectContinueTimeout: 1 * time.Second,
42                      MaxConnsPerHost:
                        ↪  cfg.GetInt("max_connections"),
43              }
44              client := &http.Client{
45                      Timeout:   timeout,
46                      Transport: transport,
47              }
48              result = func(ctx context.Context, jar http.CookieJar)
                ↪  *http.Client {
49                      if jar == nil {
50                              return client
51                      }
52                      return &http.Client{
53                              Timeout:   timeout,
54                              Jar:       jar,
55                              Transport: transport,
56                      }
57              }
58      }
59      return result
60  }
61
62  func initHTTPClient(cfg *viper.Viper) error {
63      getHTTPClient = getHTTPClientCallback(cfg.Sub("client"))
64      return nil
65  }
66
67  func GetHTTPClient(ctx context.Context, jar http.CookieJar) *http.Client
    ↪  {
68      return getHTTPClient(ctx, jar)
69  }
```

Arquivo clock.go

```
1  package config
2
3  import "github.com/fjorgemota/gurudamatricula/util/clock"
4
5  var appClock = clock.NewRealClock()
6
7  // GetClock returns the clock instance used by this application
8  func GetClock() clock.Clock {
9          return appClock
10 }
```

Arquivo closer.go

```
1  package config
2
3  var closeHandles []func() error
4
5  func addCloseHandle(close func() error) {
6          closeHandles = append(closeHandles, close)
7  }
8
9  func closeAll() error {
10         var err error
11         for _, closeFn := range closeHandles {
12                 if err == nil {
13                         err = closeFn()
14                 }
15         }
16         closeHandles = closeHandles[:0]
17         return err
18 }
```

Arquivo coder.go

```
1  package config
2
3  import (
4          "compress/zlib"
```

```
5           "errors"
6           "io"
7
8           "github.com/dsnet/compress/bzip2"
9           "github.com/fjorgemota/gurudamatricula/util/coder"
10          "github.com/spf13/viper"
11  )
12
13  type streamingCoder struct {
14          getDecoder func(reader io.Reader) coder.StreamingDecoder
15          getEncoder func(writer io.WriteCloser) coder.StreamingEncoder
16  }
17
18  var globalCoder coder.Coder
19  var globalStreamingCoder streamingCoder
20
21  func getCoderInstance(cfg *viper.Viper) (coder.Coder, error) {
22          if cfg == nil {
23                  return GetCoder(), nil
24          }
25          var cod coder.Coder
26          var err error
27          switch cfg.GetString("type") {
28          case "json":
29                  cod = coder.NewJSONCoder()
30          case "gob":
31                  cod = coder.NewGobCoder()
32          default:
33                  err = errors.New("Unrecognized streaming coding type")
34          }
35          if compressor := cfg.Sub("compressor"); compressor != nil && cod
            ↪  != nil {
36                  var handler coder.CompressorHandle
37                  switch compressor.GetString("type") {
38                  case "zlib":
39                          compressor.SetDefault("level",
                            ↪  zlib.DefaultCompression)
40                          handler =
                            ↪  coder.NewZLIBHandle(compressor.GetInt("level"))
```

```
41                  case "bzip2":
42                          compressor.SetDefault("level",
                            ↪  bzip2.DefaultCompression)
43                          handler =
                            ↪  coder.NewBZIP2Handle(compressor.GetInt("level"))
44                  default:
45                          err = errors.New("Unrecognized compressor type")
46                  }
47                  compressor.SetDefault("threshold", 1e6)
48                  cod = coder.NewCompressorCoder(cod, handler,
                    ↪  compressor.GetInt("threshold"))
49          }
50          return cod, err
51  }
52
53  func getStreamingCoderInstance(cfg *viper.Viper) (streamingCoder, error)
    ↪  {
54          var err error
55          if cfg == nil {
56                  return globalStreamingCoder, err
57          }
58          var cod streamingCoder
59          switch cfg.GetString("type") {
60          case "json":
61                  cod.getEncoder = coder.NewJSONEncoder
62                  cod.getDecoder = coder.NewJSONDecoder
63          case "gob":
64                  cod.getEncoder = coder.NewGOBStreamingEncoder
65                  cod.getDecoder = coder.NewGOBStreamingDecoder
66          default:
67                  err = errors.New("Unrecognized streaming coding type")
68          }
69          return cod, err
70  }
71
72  func initCoder(cfg *viper.Viper) error {
73          var err error
74          cfg.SetDefault("coder.type", "gob")
75          cfg.SetDefault("streaming_coder.type", "gob")
```

```
76          globalCoder, err = getCoderInstance(cfg.Sub("coder"))
77          if err == nil {
78                  globalStreamingCoder, err =
                    ↪ getStreamingCoderInstance(cfg.Sub("streaming_coder"))
79          }
80          return err
81  }
82
83  func GetCoder() coder.Coder {
84          return globalCoder
85  }
86
87  func CreateStreamingDecoder(reader io.Reader) coder.StreamingDecoder {
88          return globalStreamingCoder.getDecoder(reader)
89  }
90
91  func CreateStreamingEncoder(writer io.WriteCloser) coder.StreamingEncoder
    ↪ {
92          return globalStreamingCoder.getEncoder(writer)
93  }
```

Arquivo context.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "golang.org/x/net/context"
7   )
8
9   func GetContext(req *http.Request) context.Context {
10          return req.Context()
11  }
```

Arquivo cors.go

```
1   package config
2
```

```
3  import (
4          "net/http"
5
6          "github.com/rs/cors"
7          "github.com/spf13/viper"
8  )
9
10 var corsInstance = noopHandler
11
12 func getCorsInstance(cfg *viper.Viper) func(http.Handler) http.Handler {
13         if cfg == nil {
14                 if corsInstance != nil {
15                         return corsInstance
16                 }
17                 cfg = viper.New()
18         }
19         var options cors.Options
20         options.AllowCredentials = cfg.GetBool("allow_credentials")
21         options.AllowedHeaders =
           ↪  append(cfg.GetStringSlice("allowed_headers"), "X-CSRF-Token")
22         options.AllowedMethods = cfg.GetStringSlice("allowed_methods")
23         options.AllowedOrigins = cfg.GetStringSlice("allowed_origins")
24         options.Debug = cfg.GetBool("debug")
25         options.ExposedHeaders = cfg.GetStringSlice("exposed_headers")
26         options.MaxAge = cfg.GetInt("max_age")
27         return cors.New(options).Handler
28 }
29
30 func initCors(cfg *viper.Viper) error {
31         corsInstance = getCorsInstance(cfg.Sub("cors"))
32         return nil
33 }
34
35 func GetCors() func(http.Handler) http.Handler {
36         return corsInstance
37 }
```

Arquivo csrf.go

```
1  package config
```

```
2
3  import (
4          "errors"
5          "net/http"
6
7          "github.com/gorilla/csrf"
8          "github.com/spf13/viper"
9  )
10
11 func noopHandler(handler http.Handler) http.Handler {
12         return handler
13 }
14
15 var csrfInstance = noopHandler
16
17 func getCsrfInstance(cfg *viper.Viper) (func(http.Handler) http.Handler,
   ↪  error) {
18         if cfg == nil {
19                 cfg = viper.New()
20         }
21         cfg.SetDefault("path", "/")
22         cfg.SetDefault("enabled", false)
23         if !cfg.GetBool("enabled") {
24                 return noopHandler, nil
25         }
26         authKey := []byte(cfg.GetString("auth_key"))
27         if len(authKey) != 32 {
28                 return noopHandler, errors.New("csrf's auth key need to
                   ↪  have exactly 32 bytes")
29         }
30         var options []csrf.Option
31         if cfg.IsSet("secure") {
32                 options = append(options,
                   ↪  csrf.Secure(cfg.GetBool("secure")))
33         }
34         if cfg.IsSet("cookie_name") {
35                 options = append(options,
                   ↪  csrf.CookieName(cfg.GetString("cookie_name")))
36         }
```

```
37          if cfg.IsSet("domain") {
38                  options = append(options,
                    ↪  csrf.Domain(cfg.GetString("domain")))
39          }
40          if cfg.IsSet("field_name") {
41                  options = append(options,
                    ↪  csrf.FieldName(cfg.GetString("field_name")))
42          }
43          if cfg.IsSet("max_age") {
44                  options = append(options,
                    ↪  csrf.MaxAge(cfg.GetInt("max_age")))
45          }
46          if cfg.IsSet("path") {
47                  options = append(options,
                    ↪  csrf.Path(cfg.GetString("path")))
48          }
49          if cfg.IsSet("http_only") {
50                  options = append(options,
                    ↪  csrf.HttpOnly(cfg.GetBool("http_only")))
51          }
52          if cfg.IsSet("trusted_origins") {
53                  options = append(options,
                    ↪  csrf.TrustedOrigins(cfg.GetStringSlice("trusted_origins")))
54          }
55          return csrf.Protect(authKey, options...), nil
56  }
57
58  func initCsrf(cfg *viper.Viper) error {
59          var err error
60          csrfInstance, err = getCsrfInstance(cfg.Sub("csrf"))
61          return err
62  }
63
64  func GetCsrf() func(http.Handler) http.Handler {
65          return csrfInstance
66  }
67
68  func GetCsrfIgnorer(path string) func(http.Handler) http.Handler {
69          return func(next http.Handler) http.Handler {
```

```
70                  return http.HandlerFunc(func(resp http.ResponseWriter,
                    ↪  req *http.Request) {
71                          if req.URL.Path == path {
72                                  req = csrf.UnsafeSkipCheck(req)
73                          }
74                          next.ServeHTTP(resp, req)
75                  })
76          }
77  }
```

Arquivo dataloader_middleware.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
7           "github.com/spf13/viper"
8   )
9
10  var dataloaderMiddleware func(http.Handler) http.Handler
11
12  func getDataloaderMiddlewareInstance(cfg *viper.Viper)
    ↪  (func(http.Handler) http.Handler, error) {
13          if cfg == nil {
14                  cfg = viper.New()
15          }
16          var dependencies handlerDependencies
17          var err error
18          var result func(http.Handler) http.Handler
19          dependencies.getStore, err = getStoreCallback(cfg.Sub("store"))
20          options := dataloaders.Options{
21                  MaxBatch: cfg.GetInt("max_batch"),
22                  Wait:     cfg.GetDuration("wait"),
23          }
24          if err == nil {
25                  result = func(next http.Handler) http.Handler {
26                          return dataloaders.Middleware(next, dependencies,
                            ↪  options)
```

```
27                  }
28          }
29          return result, err
30  }
31
32  func initDataloaderMiddleware(cfg *viper.Viper) error {
33          var err error
34          dataloaderMiddleware, err =
            ↪  getDataloaderMiddlewareInstance(cfg.Sub("dataloader"))
35          return err
36  }
37
38  func getDataloaderMiddleware() func(http.Handler) http.Handler {
39          return dataloaderMiddleware
40  }
```

Arquivo ddiffer.go

```
1   package config
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
5           "github.com/spf13/viper"
6   )
7
8   var ddifferHandler ddiffer.PipelineHandler
9
10  func getDDifferLimit(cfg *viper.Viper) ddiffer.Limit {
11          if cfg == nil {
12                  cfg = viper.New()
13          }
14          return ddiffer.Limit{
15                  Items: cfg.GetInt("items"),
16                  Size:  cfg.GetInt("size"),
17          }
18  }
19
20  func getDDifferHandler(cfg *viper.Viper) (ddiffer.PipelineHandler, error)
    ↪  {
21          if cfg == nil {
```

```
22                  cfg = viper.New()
23          }
24          var err error
25          var handler ddiffer.PipelineHandler
26          var dependencies handlerDependencies
27          dependencies.getManager, err =
            ↪  getManagerCallback(cfg.Sub("manager"))
28          dependencies.streaming, err =
            ↪  getStreamingCoderInstance(cfg.Sub("streaming_coder"))
29          if err == nil {
30                  options := ddiffer.PipelineGlobalOptions{
31                          ComparersLimits:
                            ↪  getDDifferLimit(cfg.Sub("comparer")),
32                          ConvertersLimits:
                            ↪  getDDifferLimit(cfg.Sub("converter")),
33                  }
34                  handler, err =
                    ↪  ddiffer.NewPipelineHandler(GetPipelineDispatcher(),
                    ↪  dependencies, options)
35          }
36          return handler, err
37  }
38
39  func initDDifferHandler(cfg *viper.Viper) error {
40          var err error
41          ddifferHandler, err = getDDifferHandler(cfg.Sub("ddiffer"))
42          return err
43  }
44
45  func GetDDifferHandler() ddiffer.PipelineHandler {
46          return ddifferHandler
47  }
```

Arquivo fetcher.go

```
1   package config
2
3   import (
4           "net/http"
5
```

```
6         "github.com/fjorgemota/gurudamatricula/controllers"
7
8         "github.com/fjorgemota/gurudamatricula/robot/fetcher"
9         "github.com/spf13/viper"
10 )
11
12 var fetcherController http.Handler
13
14 func getFetcherControllerInstance(cfg *viper.Viper) (http.Handler, error)
   ↪ {
15         if cfg == nil {
16                 cfg = viper.New()
17         }
18         var err error
19         dispatcher := GetPipelineDispatcher()
20         var dependencies handlerDependencies
21         dependencies.getStore, err = getStoreCallback(cfg.Sub("store"))
22         if err == nil {
23                 dependencies.getManager, err =
                   ↪ getManagerCallback(cfg.Sub("manager"))
24         }
25         if err == nil {
26                 dependencies.getIndex, err =
                   ↪ getIndexCallback(cfg.Sub("index"))
27         }
28         if err == nil {
29                 dependencies.getLogger, err =
                   ↪ getLoggerCallback(cfg.Sub("log"))
30         }
31         if err == nil {
32                 dependencies.streaming, err =
                   ↪ getStreamingCoderInstance(cfg.Sub("streaming_coder"))
33         }
34         if err == nil {
35                 dependencies.getHTTPClient =
                   ↪ getHTTPClientCallback(cfg.Sub("client"))
36         }
37         var instance fetcher.Fetcher
38         var handler http.Handler
```

```
39          if err == nil {
40                  var options fetcher.Options
41                  options.Tasks.Lock = cfg.GetString("tasks.lock")
42                  options.Tasks.Monitor = cfg.GetString("tasks.monitor")
43                  options.Tasks.Fetch = cfg.GetString("tasks.fetch")
44                  options.Tasks.Compare = cfg.GetString("tasks.compare")
45                  options.Tasks.Import = cfg.GetString("tasks.import")
46                  options.Tasks.Optimize = cfg.GetString("tasks.optimize")
47                  options.Tasks.Finalize = cfg.GetString("tasks.finalize")
48                  options.DelayCheck = cfg.GetDuration("delay_check")
49                  options.Parallel = cfg.GetBool("parallel")
50                  options.LimitQueue = cfg.GetInt("limit_queue")
51                  options.LimitVersions = cfg.GetInt("limit_versions")
52                  options.Clock = GetClock()
53                  instance, err = fetcher.NewFetcher(dispatcher,
                    ↪  dependencies, options)
54          }
55          if err == nil {
56                  handler = controllers.Fetcher(dependencies, instance)
57          }
58          return handler, err
59  }
60
61  func initFetcher(cfg *viper.Viper) error {
62          var err error
63          fetcherController, err =
            ↪  getFetcherControllerInstance(cfg.Sub("fetcher"))
64          return err
65  }
66
67  func GetFetcherController() http.Handler {
68          return fetcherController
69  }
```

Arquivo file.go

```
1   package config
2
3   import (
4           "net/http"
```

```
5
6           "github.com/fjorgemota/gurudamatricula/controllers"
7
8           "cloud.google.com/go/storage"
9           "github.com/fjorgemota/gurudamatricula/util/file"
10          "github.com/spf13/viper"
11          "golang.org/x/net/context"
12  )
13
14  var getManager func(ctx context.Context) (file.Manager, error)
15  var managerRouter *http.ServeMux
16
17  func getManagerCallback(cfg *viper.Viper) (func(ctx context.Context)
    ↪  (file.Manager, error), error) {
18          if cfg == nil {
19                  return GetManager, nil
20          }
21          var instance file.Manager
22          var err error
23          switch cfg.GetString("type") {
24          case "gcs":
25                  basePath := cfg.GetString("base_path")
26                  bucketName := cfg.GetString("bucket_name")
27                  return func(ctx context.Context) (file.Manager, error) {
28                          var result file.Manager
29                          client, localErr := storage.NewClient(ctx)
30                          if localErr == nil {
31                                  bucket := client.Bucket(bucketName)
32                                  result = file.NewGCSManager(basePath, ctx,
                                    ↪  bucket)
33                          }
34                          return result, localErr
35                  }, nil
36          case "memory":
37                  instance = file.NewMemoryManager()
38          case "local":
39                  instance, err =
                    ↪  file.NewLocalManager(cfg.GetString("base_path"))
40          }
```

```
41          result := func(ctx context.Context) (file.Manager, error) {
42                  return instance, err
43          }
44          if cfg.IsSet("public_path") {
45                  dependencies := handlerDependencies{
46                          getManager: result,
47                  }
48                  controller := controllers.FileController(dependencies)
49                  managerRouter.Handle(cfg.GetString("public_path"),
                    ↪  controller)
50          }
51          return result, err
52  }
53
54  func initManager(cfg *viper.Viper) error {
55          var err error
56          cfg.SetDefault("manager.type", "memory")
57          managerRouter = http.NewServeMux()
58          getManager, err = getManagerCallback(cfg.Sub("manager"))
59          return err
60  }
61
62  func GetManager(ctx context.Context) (file.Manager, error) {
63          return getManager(ctx)
64  }
65
66  func getManagerRouter() http.Handler {
67          return managerRouter
68  }
```

Arquivo graphql_controller.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/fjorgemota/gurudamatricula/controllers"
7
8           "github.com/spf13/viper"
```

```
9  )
10
11 var graphqlController http.Handler
12
13 func getGraphQLControllerInstance(cfg *viper.Viper) (http.Handler, error)
   ↪ {
14         if cfg == nil {
15                 if graphqlController != nil {
16                         return graphqlController, nil
17                 }
18                 cfg = viper.New()
19         }
20         var dependencies handlerDependencies
21         var err error
22         var result http.Handler
23         dependencies.getStore, err = getStoreCallback(cfg.Sub("store"))
24         if err == nil {
25                 dependencies.getIndex, err =
                   ↪  getIndexCallback(cfg.Sub("index"))
26         }
27         if err == nil {
28                 result = controllers.GraphQLController(dependencies)
29         }
30         return result, err
31 }
32
33 func initGraphQLController(cfg *viper.Viper) error {
34         var err error
35         graphqlController, err =
           ↪  getGraphQLControllerInstance(cfg.Sub("graphql"))
36         return err
37 }
38
39 func getGraphQLController() http.Handler {
40         return graphqlController
41 }
42
43 func getGraphQLPlaygroundController() http.Handler {
```

```
44          return
            ↪  controllers.GraphQLPlaygroundController(handlerDependencies{})
45  }
```

Arquivo gzip.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/NYTimes/gziphandler"
7           "github.com/spf13/viper"
8   )
9
10  var gzipInstance = noopHandler
11
12  func getGzipInstance(cfg *viper.Viper) (func(http.Handler) http.Handler,
    ↪  error) {
13          if cfg == nil {
14                  if gzipInstance != nil {
15                          return gzipInstance, nil
16                  }
17                  cfg = viper.New()
18                  cfg.SetDefault("level", 9)
19          }
20          return gziphandler.NewGzipLevelAndMinSize(cfg.GetInt("level"),
            ↪  cfg.GetInt("min_size"))
21  }
22
23  func initGzip(cfg *viper.Viper) error {
24          var err error
25          gzipInstance, err = getGzipInstance(cfg.Sub("gzip"))
26          return err
27  }
28
29  func GetGzip() func(http.Handler) http.Handler {
30          return gzipInstance
31  }
```

Arquivo handler.go

```
1  package config
2
3  import (
4          "io"
5          "net/http"
6          "net/http/cookiejar"
7
8          "github.com/fjorgemota/gurudamatricula/util/clock"
9          "github.com/fjorgemota/gurudamatricula/util/middlewares"
10         "github.com/fjorgemota/gurudamatricula/util/pipeline"
11         "github.com/fjorgemota/gurudamatricula/util/taskqueue"
12
13         "github.com/sirupsen/logrus"
14         "github.com/volatiletech/authboss"
15
16         "golang.org/x/net/publicsuffix"
17
18         "github.com/fjorgemota/gurudamatricula/robot/util/ccookiejar"
19
20         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
21         "github.com/fjorgemota/gurudamatricula/robot/importer"
22         "github.com/fjorgemota/gurudamatricula/util/coder"
23         "github.com/fjorgemota/gurudamatricula/util/file"
24         "github.com/fjorgemota/gurudamatricula/util/indexer"
25         "github.com/fjorgemota/gurudamatricula/util/kv"
26         "golang.org/x/net/context"
27 )
28
29 type handlerDependencies struct {
30         streaming     streamingCoder
31         getLogger     func(ctx context.Context) logrus.FieldLogger
32         getManager    func(ctx context.Context) (file.Manager, error)
33         getStore      func(ctx context.Context) (kv.Store, error)
34         getIndex      func(ctx context.Context, data importer.IndexData)
          ↪  (indexer.Index, error)
35         getHTTPClient func(ctx context.Context, jar http.CookieJar)
          ↪  *http.Client
```

```
36          getDDiffer    func() ddiffer.PipelineHandler
37          getImporter   func() importer.PipelineHandler
38  }
39
40  func (ddd handlerDependencies) GetLogger(ctx context.Context)
    ↪  logrus.FieldLogger {
41          return ddd.getLogger(ctx)
42  }
43
44  func (ddd handlerDependencies) GetContext(req *http.Request)
    ↪  context.Context {
45          return GetContext(req)
46  }
47
48  func (ddd handlerDependencies) GetRequest(ctx context.Context)
    ↪  *http.Request {
49          return middlewares.GetRequest(ctx)
50  }
51
52  func (ddd handlerDependencies) GetClock() clock.Clock {
53          return GetClock()
54  }
55
56  func (ddd handlerDependencies) GetAuthBoss() *authboss.Authboss {
57          return GetAuthBoss()
58  }
59
60  func (ddd handlerDependencies) GetRoute(name string) string {
61          return GetRoute(name)
62  }
63
64  func (ddd handlerDependencies) GetPipeline(ctx context.Context)
    ↪  (pipeline.Pipeline, error) {
65          return GetPipeline(ctx)
66  }
67
68  func (ddd handlerDependencies) GetCookieJar() (ccookiejar.CodedCookieJar,
    ↪  error) {
69          return ccookiejar.NewCookieJarWithOptions(&cookiejar.Options{
```

```
70                  PublicSuffixList: publicsuffix.List,
71          })
72  }
73
74  func (ddd handlerDependencies) GetStore(ctx context.Context) (kv.Store,
    ↪  error) {
75          return ddd.getStore(ctx)
76  }
77
78  func (ddd handlerDependencies) GetManager(ctx context.Context)
    ↪  (file.Manager, error) {
79          return ddd.getManager(ctx)
80  }
81
82  func (ddd handlerDependencies) GetStreamingDecoder(reader io.Reader)
    ↪  coder.StreamingDecoder {
83          return ddd.streaming.getDecoder(reader)
84  }
85
86  func (ddd handlerDependencies) GetStreamingEncoder(writer io.WriteCloser)
    ↪  coder.StreamingEncoder {
87          return ddd.streaming.getEncoder(writer)
88  }
89
90  func (ddd handlerDependencies) GetIndex(ctx context.Context, data
    ↪  importer.IndexData) (indexer.Index, error) {
91          return ddd.getIndex(ctx, data)
92  }
93
94  func (ddd handlerDependencies) GetDDiffer() ddiffer.PipelineHandler {
95          return GetDDifferHandler()
96  }
97
98  func (ddd handlerDependencies) GetImporter() importer.PipelineHandler {
99          return GetImporterHandler()
100 }
101
102 func (ddd handlerDependencies) GetTaskQueueCaller(ctx context.Context)
    ↪  taskqueue.TaskCaller {
```

```
103         return GetTaskQueueCaller(ctx)
104 }
105
106 func (ddd handlerDependencies) GetHTTPClient(ctx context.Context, jar
    ↪ http.CookieJar) *http.Client {
107         return ddd.getHTTPClient(ctx, jar)
108 }
```

Arquivo importer.go

```
1  package config
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/robot/importer"
5          "github.com/spf13/viper"
6  )
7
8  var importerHandler importer.PipelineHandler
9
10 func getImporterHandler(cfg *viper.Viper) (importer.PipelineHandler,
   ↪ error) {
11         if cfg == nil {
12                 cfg = viper.New()
13         }
14         var err error
15         var handler importer.PipelineHandler
16         var dependencies handlerDependencies
17         dependencies.getStore, err = getStoreCallback(cfg.Sub("store"))
18         if err == nil {
19                 dependencies.getManager, err =
                   ↪ getManagerCallback(cfg.Sub("manager"))
20         }
21         if err == nil {
22                 dependencies.streaming, err =
                   ↪ getStreamingCoderInstance(cfg.Sub("streaming_coder"))
23         }
24         if err == nil {
25                 dependencies.getIndex = GetIndex
26         }
27         var options importer.PipelineOptions
```

```
28          if err == nil {
29                  options.Parallel = cfg.GetBool("parallel")
30                  options.TaskNames.DeleteDataset =
                    ↪  cfg.GetString("tasks.delete_dataset")
31                  options.TaskNames.ImportChangedDataset =
                    ↪  cfg.GetString("tasks.import_changed")
32                  options.TaskNames.ImportDataset =
                    ↪  cfg.GetString("tasks.import_dataset")
33                  options.TaskNames.ImportIndex =
                    ↪  cfg.GetString("tasks.import_index")
34                  options.TaskNames.OptimizeIndex =
                    ↪  cfg.GetString("tasks.optimize_index")
35                  handler, err =
                    ↪  importer.NewPipelineHandler(GetPipelineDispatcher(),
                    ↪  dependencies, options)
36          }
37          return handler, err
38  }
39
40  func initImporterHandler(cfg *viper.Viper) error {
41          var err error
42          importerHandler, err = getImporterHandler(cfg.Sub("importer"))
43          return err
44  }
45
46  func GetImporterHandler() importer.PipelineHandler {
47          return importerHandler
48  }
```

Arquivo index.go

```
1   package config
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/robot/importer"
5           "github.com/fjorgemota/gurudamatricula/search"
6           "github.com/fjorgemota/gurudamatricula/util/indexer"
7           "github.com/fjorgemota/gurudamatricula/util/kv"
8           "github.com/spf13/viper"
9           "golang.org/x/net/context"
```

```
10 )
11
12 var getIndex func(ctx context.Context, data importer.IndexData)
   ↪ (indexer.Index, error)
13
14 func getIndexCallback(cfg *viper.Viper) (func(ctx context.Context, data
   ↪ importer.IndexData) (indexer.Index, error), error) {
15         if cfg == nil {
16                 if getIndex != nil {
17                         return GetIndex, nil
18                 }
19                 cfg = viper.New()
20         }
21         cfg.SetDefault("controller.type", "shard")
22         cfg.SetDefault("controller.table_prefix", "indexes")
23         cfg.SetDefault("controller.maximum_size", 9e5)
24         cfg.SetDefault("exact.default_field", "default")
25         cfg.SetDefault("exact.false_positive_rate", 0.01)
26         cfg.SetDefault("exact.merge_min_versions", 3)
27         cfg.SetDefault("exact.meta_table", "meta")
28         cfg.SetDefault("fulltext.max_distance", 5)
29         cfg.SetDefault("fulltext.max_tree_distance", 3)
30         cfg.SetDefault("fulltext.max_user_distance", 3)
31         storer, err := getStoreCallback(cfg.Sub("store"))
32         var controller func(store kv.Store) indexer.Controller
33         switch cfg.GetString("controller.type") {
34         case "kv":
35                 var options indexer.KVOptions
36                 options.TablePrefix =
                   ↪ cfg.GetString("controller.table_prefix")
37                 if err == nil {
38                         options.Coder, err =
                           ↪ getCoderInstance(cfg.Sub("controller.coder"))
39                 }
40                 controller = func(store kv.Store) indexer.Controller {
41                         return indexer.NewKVController(store, options)
42                 }
43         case "shard":
44                 var options indexer.ShardKVOptions
```

```
45                options.MaximumSize =
                  ↪  cfg.GetInt("controller.maximum_size")
46                options.TablePrefix =
                  ↪  cfg.GetString("controller.table_prefix")
47                var cod streamingCoder
48                cod, err =
                  ↪  getStreamingCoderInstance(cfg.Sub("controller.streaming_coder"))
49                dependencies := handlerDependencies{streaming: cod}
50                controller = func(store kv.Store) indexer.Controller {
51                        return indexer.NewShardKVController(store,
                          ↪  dependencies, options)
52                }
53        }
54        var exactOptions indexer.ExactOptions
55        exactOptions.DefaultField = cfg.GetString("exact.default_field")
56        exactOptions.FalsePositiveKeys =
          ↪  cfg.GetFloat64("exact.false_positive_rate")
57        exactOptions.AutoMergeMinVersions =
          ↪  cfg.GetInt("exact.auto_merge_min_versions")
58        exactOptions.MergeMinVersions =
          ↪  cfg.GetInt("exact.merge_min_versions")
59        exactOptions.MetaTable = cfg.GetString("exact.meta_table")
60        var fullTextOptions indexer.FullTextOptions
61        fullTextOptions.IsExact = search.IsExact
62        fullTextOptions.MaxDistance =
          ↪  cfg.GetUint("fulltext.max_distance")
63        fullTextOptions.MaxTreeDistance =
          ↪  cfg.GetUint("fulltext.max_tree_distance")
64        fullTextOptions.MaxUserDistance =
          ↪  cfg.GetUint("fulltext.max_user_distance")
65        return func(ctx context.Context, data importer.IndexData)
          ↪  (indexer.Index, error) {
66                var index indexer.Index
67                var store kv.Store
68                localErr := err
69                if localErr == nil {
70                        store, localErr = storer(ctx)
71                }
72                if localErr == nil {
```

```
73                          control := controller(store)
74                          options := exactOptions
75                          options.Name += data.ParentID + "-" +
                            ↪  data.Target.String()
76                          index = indexer.NewFullTextIndexer(store, control,
                            ↪  options, fullTextOptions)
77                  }
78                  return index, localErr
79          }, err
80  }
81
82  func initIndex(cfg *viper.Viper) error {
83          var err error
84          getIndex, err = getIndexCallback(cfg.Sub("index"))
85          return err
86  }
87
88  func GetIndex(ctx context.Context, data importer.IndexData)
    ↪  (indexer.Index, error) {
89          return getIndex(ctx, data)
90  }
```

Arquivo kv.go

```
1   // +build !appengine
2
3   package config
4
5   import (
6           "errors"
7           "fmt"
8           "os"
9           "time"
10
11          "cloud.google.com/go/datastore"
12          "github.com/boltdb/bolt"
13          "github.com/dgraph-io/badger/v2"
14          "github.com/dgraph-io/badger/v2/options"
15          "github.com/fjorgemota/gurudamatricula/util/cache"
16          "github.com/fjorgemota/gurudamatricula/util/coder"
```

```
17          "github.com/fjorgemota/gurudamatricula/util/file"
18          "github.com/fjorgemota/gurudamatricula/util/kv"
19          "github.com/spf13/viper"
20          "go.etcd.io/bbolt"
21          "golang.org/x/net/context"
22  )
23
24  var getStore func(ctx context.Context) (kv.Store, error)
25
26  func getBadgerOptions(cfg *viper.Viper) badger.Options {
27          opts := badger.DefaultOptions(cfg.GetString("options.path"))
28          if cfg.IsSet("options.logging") && cfg.GetBool("options.logging")
            ↪  == false {
29                  opts = opts.WithLogger(nil)
30          }
31          if cfg.IsSet("options.path_value") {
32                  opts =
                    ↪  opts.WithValueDir(cfg.GetString("options.path_value"))
33          }
34          if cfg.IsSet("options.sync_writes") {
35                  opts =
                    ↪  opts.WithSyncWrites(cfg.GetBool("options.sync_writes"))
36          }
37          switch cfg.GetString("options.table_loading_mode") {
38          case "mmap":
39                  opts = opts.WithTableLoadingMode(options.MemoryMap)
40          case "file":
41                  opts = opts.WithTableLoadingMode(options.FileIO)
42          case "ram":
43                  opts = opts.WithTableLoadingMode(options.LoadToRAM)
44          }
45          switch cfg.GetString("options.value_log_loading_mode") {
46          case "mmap":
47                  opts = opts.WithValueLogLoadingMode(options.MemoryMap)
48          case "file":
49                  opts = opts.WithValueLogLoadingMode(options.FileIO)
50          case "ram":
51                  opts = opts.WithValueLogLoadingMode(options.LoadToRAM)
52          }
```

```
53          if cfg.IsSet("options.num_versions_to_keep") {
54                  opts =
                    ↪  opts.WithNumVersionsToKeep(cfg.GetInt("options.num_versions_to_
55          }
56          if cfg.IsSet("options.read_only") {
57                  opts =
                    ↪  opts.WithReadOnly(cfg.GetBool("options.read_only"))
58          }
59          if cfg.IsSet("options.truncate") {
60                  opts = opts.WithTruncate(cfg.GetBool("options.truncate"))
61          }
62          if cfg.IsSet("options.read_only") {
63                  opts =
                    ↪  opts.WithReadOnly(cfg.GetBool("options.read_only"))
64          }
65          if cfg.IsSet("options.max_table_size") {
66                  opts =
                    ↪  opts.WithMaxTableSize(cfg.GetInt64("options.max_table_size"))
67          }
68          if cfg.IsSet("options.level_size_multiplier") {
69                  opts =
                    ↪  opts.WithLevelSizeMultiplier(cfg.GetInt("options.level_size_mul
70          }
71          if cfg.IsSet("options.max_levels") {
72                  opts =
                    ↪  opts.WithMaxLevels(cfg.GetInt("options.max_levels"))
73          }
74          if cfg.IsSet("options.value_threshold") {
75                  opts =
                    ↪  opts.WithValueThreshold(cfg.GetInt("options.value_threshold"))
76          }
77          if cfg.IsSet("options.num_memtables") {
78                  opts =
                    ↪  opts.WithNumMemtables(cfg.GetInt("options.num_memtables"))
79          }
80          if cfg.IsSet("options.num_level_zero_tables") {
81                  opts =
                    ↪  opts.WithNumLevelZeroTables(cfg.GetInt("options.num_level_zero_
82          }
```

```
83          if cfg.IsSet("options.num_level_zero_tables_stall") {
84                  opts =
                    ↪  opts.WithNumLevelZeroTablesStall(cfg.GetInt("options.num_level_zero_
85          }
86          if cfg.IsSet("options.level_one_size") {
87                  opts =
                    ↪  opts.WithLevelOneSize(cfg.GetInt64("options.level_one_size"))
88          }
89          if cfg.IsSet("options.value_log_file_size") {
90                  opts =
                    ↪  opts.WithValueLogFileSize(cfg.GetInt64("options.value_log_file_size'
91          }
92          if cfg.IsSet("options.value_log_max_entries") {
93                  opts =
                    ↪  opts.WithValueLogMaxEntries(cfg.GetUint32("options.value_log_max_ent
94          }
95          if cfg.IsSet("options.num_compactors") {
96                  opts =
                    ↪  opts.WithNumCompactors(cfg.GetInt("options.num_compactors"))
97          }
98          if cfg.IsSet("options.compact_l0_on_close") {
99                  opts =
                    ↪  opts.WithCompactL0OnClose(cfg.GetBool("options.compact_l0_on_close")
100         }
101         if cfg.IsSet("options.log_rotates_to_flush") {
102                 opts =
                    ↪  opts.WithLogRotatesToFlush(cfg.GetInt32("options.log_rotates_to_flus
103         }
104         return opts
105 }
106
107 func gcBadger(ticker *time.Ticker, stop chan struct{}, handle *badger.DB,
    ↪  discardRatio float64) {
108         for {
109                 select {
110                 case <-ticker.C:
111                 again:
112                         fmt.Println("[badger] Running GC")
113                         err := handle.RunValueLogGC(discardRatio)
```

```
114                         if err == nil {
115                                 goto again
116                         }
117                 case <-stop:
118                         fmt.Println("[badger] Stopping GC goroutine")
119                         return
120                 }
121         }
122 }
123
124 func getStoreCallback(cfg *viper.Viper) (func(ctx context.Context)
    ↪  (kv.Store, error), error) {
125         if cfg == nil {
126                 return GetStore, nil
127         }
128         var instance kv.Store
129         var err error
130         switch cfg.GetString("type") {
131         default:
132                 err = errors.New("Unrecognized database type")
133         case "memory":
134                 instance = kv.NewMemoryStore()
135         case "bbolt":
136                 var handle *bbolt.DB
137                 var cod coder.Coder
138                 cfg.SetDefault("options.path", "./db")
139                 cfg.SetDefault("options.permission", 0600)
140                 cod, err = getCoderInstance(cfg.Sub("coder"))
141                 if err == nil {
142                         handle, err =
                        ↪  bbolt.Open(cfg.GetString("options.path"),
                        ↪  os.FileMode(cfg.GetInt32("options.permission")),
                        ↪  &bbolt.Options{
143                                 InitialMmapSize:
                                ↪  cfg.GetInt("options.initial_mmap_size"),
144                                 MmapFlags:
                                ↪  cfg.GetInt("options.mmap_flags"),
145                                 NoGrowSync:
                                ↪  cfg.GetBool("options.no_grow_sync"),
```

```
146                                 NoFreelistSync:
                                    ↪  cfg.GetBool("options.no_freelist_sync"),
147                                 NoSync:
                                    ↪  cfg.GetBool("options.no_sync"),
148                                 ReadOnly:
                                    ↪  cfg.GetBool("options.read_only"),
149                                 Timeout:
                                    ↪  cfg.GetDuration("options.timeout"),
150                                 PageSize:
                                    ↪  cfg.GetInt("options.page_size"),
151                                 FreelistType:
                                    ↪  bbolt.FreelistType(cfg.GetString("options.freelist_t
152                         })
153                 }
154                 if err == nil {
155                         addCloseHandle(handle.Close)
156                         instance = kv.NewBBoltStore(cod, handle)
157                 }
158         case "bolt":
159                 var handle *bolt.DB
160                 var cod coder.Coder
161                 cfg.SetDefault("options.path", "./db")
162                 cfg.SetDefault("options.permission", 0600)
163                 cod, err = getCoderInstance(cfg.Sub("coder"))
164                 if err == nil {
165                         handle, err =
                         ↪  bolt.Open(cfg.GetString("options.path"),
                         ↪  os.FileMode(cfg.GetInt32("options.permission")),
                         ↪  &bolt.Options{
166                                 InitialMmapSize:
                                    ↪  cfg.GetInt("options.initial_mmap_size"),
167                                 MmapFlags:
                                    ↪  cfg.GetInt("options.mmap_flags"),
168                                 NoGrowSync:
                                    ↪  cfg.GetBool("options.no_grow_sync"),
169                                 ReadOnly:
                                    ↪  cfg.GetBool("options.read_only"),
170                                 Timeout:
                                    ↪  cfg.GetDuration("options.timeout"),
```

```
171                         })
172                 }
173                 if err == nil {
174                         addCloseHandle(handle.Close)
175                         instance = kv.NewBoltStore(cod, handle)
176                 }
177         case "badger":
178                 var handle *badger.DB
179                 var cod coder.Coder
180                 cfg.SetDefault("options.path", "./db")
181                 cfg.SetDefault("options.delay", "3s")
182                 cfg.SetDefault("options.gc_interval", "10m")
183                 cfg.SetDefault("options.gc_discard_ratio", "0.7")
184                 cfg.SetDefault("options.retries", "10")
185                 opts := getBadgerOptions(cfg)
186                 handle, err = badger.Open(opts)
187                 cod, err := getCoderInstance(cfg.Sub("coder"))
188                 if err == nil {
189                         ticker :=
                        ↪  time.NewTicker(cfg.GetDuration("options.gc_interval"))
190                         stop := make(chan struct{})
191                         go gcBadger(ticker, stop, handle,
                        ↪  cfg.GetFloat64("options.gc_discard_ratio"))
192                         addCloseHandle(func() error {
193                                 close(stop)
194                                 return nil
195                         })
196                         addCloseHandle(handle.Close)
197                         badgerOptions := kv.BadgerOptions{
198                                 Retries: cfg.GetInt("options.retries"),
199                                 Delay:
                                ↪  cfg.GetDuration("options.delay"),
200                                 Clock:   GetClock(),
201                         }
202                         instance = kv.NewBadgerStoreWithOptions(cod,
                        ↪  handle, badgerOptions)
203                 }
204         case "file":
205                 var cod coder.Coder
```

```
206             var getManager func(ctx context.Context) (file.Manager,
                ↪  error)
207             getManager, err = getManagerCallback(cfg.Sub("manager"))
208             if err == nil {
209                     cod, err = getCoderInstance(cfg.Sub("coder"))
210             }
211             return func(ctx context.Context) (kv.Store, error) {
212                     var result kv.Store
213                     manager, localErr := getManager(ctx)
214                     if localErr == nil {
215                             result = kv.NewFileStore(manager, cod)
216                     }
217                     return result, localErr
218             }, err
219     case "cached":
220             var storer func(ctx context.Context) (kv.Store, error)
221             cacher, err := getCacheCallback(cfg.Sub("cache"))
222             if err == nil {
223                     storer, err = getStoreCallback(cfg.Sub("store"))
224             }
225             return func(ctx context.Context) (kv.Store, error) {
226                     var result kv.Store
227                     var cac cache.Cache
228                     store, localErr := storer(ctx)
229                     if localErr == nil {
230                             cac, localErr = cacher(ctx)
231                     }
232                     if localErr == nil {
233                             result = kv.NewCachedStore(store, cac)
234                     }
235                     return result, localErr
236             }, err
237     case "appengine":
238             return func(ctx context.Context) (kv.Store, error) {
239                     return kv.NewAppEngineStore(ctx), nil
240             }, err
241
242     case "datastore":
```

```
243                 handler, err := datastore.NewClient(context.Background(),
                    ↪  cfg.GetString("project_id"))
244                 if err == nil {
245                         addCloseHandle(handler.Close)
246                 }
247                 return func(ctx context.Context) (kv.Store, error) {
248                         var result kv.Store
249                         localErr := err
250                         if localErr == nil {
251                                 result = kv.NewCloudDataStore(handler,
                                    ↪  ctx)
252                         }
253                         return result, err
254                 }, err
255
256         }
257         return func(ctx context.Context) (kv.Store, error) {
258                 return instance, err
259         }, err
260 }
261
262 func initStore(cfg *viper.Viper) error {
263         cfg.SetDefault("store.type", "memory")
264         var err error
265         getStore, err = getStoreCallback(cfg.Sub("store"))
266         return err
267 }
268
269 func GetStore(ctx context.Context) (kv.Store, error) {
270         return getStore(ctx)
271 }
```

Arquivo logger.go

```
1  package config
2
3  import (
4          "io"
5          "io/ioutil"
6          "os"
```

```
7
8           "golang.org/x/net/context"
9
10          "github.com/fjorgemota/gurudamatricula/util/aehook"
11          "github.com/sirupsen/logrus"
12          "github.com/spf13/viper"
13  )
14
15  var getLogger func(ctx context.Context) logrus.FieldLogger
16
17  func getLoggerOutput(output string) (io.Writer, error) {
18          var out io.Writer
19          var err error
20          switch output {
21          case "stdout":
22                  out = os.Stdout
23          case "stderr":
24                  out = os.Stderr
25          case "discard":
26                  out = ioutil.Discard
27          default:
28                  out, err = os.Create(output)
29          }
30          return out, err
31  }
32
33  func getLoggerCallback(cfg *viper.Viper) (func(ctx context.Context)
    ↪  logrus.FieldLogger, error) {
34          if cfg == nil && getLogger != nil {
35                  return GetLogger, nil
36          }
37          hooks := cfg.GetStringSlice("hooks")
38          out, err := getLoggerOutput(cfg.GetString("output"))
39          var level logrus.Level
40          if err == nil {
41                  level, err = logrus.ParseLevel(cfg.GetString("level"))
42          }
43          return func(ctx context.Context) logrus.FieldLogger {
44                  logger := logrus.New()
```

```
45                  logger.Out = out
46                  logger.Level = level
47                  for _, hook := range hooks {
48                          switch hook {
49                          case "appengine":
50                                  logger.Hooks.Add(aehook.NewAEHook(ctx))
51                          }
52                  }
53                  return logger
54          }, err
55  }
56
57  func initLogger(cfg *viper.Viper) error {
58          cfg.SetDefault("log.output", "stderr")
59          cfg.SetDefault("log.level", "debug")
60          var err error
61          getLogger, err = getLoggerCallback(cfg.Sub("log"))
62          return err
63  }
64
65  // GetLogger returns a logger to the system, based on a context
66  func GetLogger(ctx context.Context) logrus.FieldLogger {
67          return getLogger(ctx)
68  }
```

Arquivo pipeline.go

```
1   package config
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/util/clock"
5           "github.com/fjorgemota/gurudamatricula/util/coder"
6           "github.com/fjorgemota/gurudamatricula/util/kv"
7           "github.com/fjorgemota/gurudamatricula/util/pipeline"
8           "github.com/spf13/viper"
9           "golang.org/x/net/context"
10  )
11
12  var pipelineDispatcher pipeline.Dispatcher
13  var getPipeline func(ctx context.Context) (pipeline.Pipeline, error)
```

```
14
15 func getPipelineDispatcherInstance(cfg *viper.Viper) (pipeline.Dispatcher,
   ↪  error) {
16         if cfg == nil {
17                 cfg = viper.New()
18         }
19         var result pipeline.Dispatcher
20         cod, err := getCoderInstance(cfg.Sub("coder"))
21         if err == nil {
22                 result = pipeline.NewDispatcher(cod)
23         }
24         return result, err
25 }
26
27 func getPipelineCallback(dispatcher pipeline.Dispatcher, cfg
   ↪  *viper.Viper) (func(ctx context.Context) (pipeline.Pipeline, error),
   ↪  error) {
28         if cfg == nil {
29                 cfg = viper.New()
30         }
31         var cod coder.Coder
32         getLogger, err := getLoggerCallback(cfg.Sub("log"))
33         if err == nil {
34                 cod, err = getCoderInstance(cfg.Sub("coder"))
35         }
36         var getStore func(ctx context.Context) (kv.Store, error)
37         if err == nil {
38                 getStore, err = getStoreCallback(cfg.Sub("database"))
39         }
40         clk := clock.NewRealClock()
41         options := pipeline.GlobalOptions{
42                 AbortTaskName:      cfg.GetString("tasks.abort"),
43                 ParentDoneTaskName: cfg.GetString("tasks.parent_done"),
44                 WorkTaskName:       cfg.GetString("tasks.work"),
45                 MaxAttempts:        cfg.GetInt("max_attempts"),
46                 Table:              cfg.GetString("table"),
47         }
48         if err == nil {
49                 var workerHandler handlerDependencies
```

```
50                  workerHandler.getStore = getStore
51                  workerHandler.getLogger = getLogger
52                  taskDispatcher := GetTaskQueueDispatcher()
53                  err = pipeline.RegisterWorker(taskDispatcher, dispatcher,
                    ↪  cod, clk, workerHandler, options)
54          }
55          return func(ctx context.Context) (pipeline.Pipeline, error) {
56                  var result pipeline.Pipeline
57                  var store kv.Store
58                  caller := GetTaskQueueCaller(ctx)
59                  logger := getLogger(ctx)
60                  localErr := err
61                  if localErr == nil {
62                          store, localErr = getStore(ctx)
63                  }
64                  if localErr == nil {
65                          result = pipeline.NewPipeline(logger, cod, clk,
                            ↪  caller, store, dispatcher, options)
66                  }
67                  return result, localErr
68          }, err
69  }
70
71  func initPipeline(cfg *viper.Viper) error {
72          var err error
73          subcfg := cfg.Sub("pipeline")
74          pipelineDispatcher, err = getPipelineDispatcherInstance(subcfg)
75          if err == nil {
76                  getPipeline, err = getPipelineCallback(pipelineDispatcher,
                    ↪  subcfg)
77          }
78          return err
79  }
80
81  func GetPipelineDispatcher() pipeline.Dispatcher {
82          return pipelineDispatcher
83  }
84
85  func GetPipeline(ctx context.Context) (pipeline.Pipeline, error) {
```

```
86          return getPipeline(ctx)
87  }
```

Arquivo redirector_controller.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/fjorgemota/gurudamatricula/controllers"
7
8           "github.com/spf13/viper"
9   )
10
11  var redirectorController http.Handler = http.NotFoundHandler()
12
13  func getRedirectorInstance(cfg *viper.Viper) (http.Handler, error) {
14          if cfg == nil || !cfg.IsSet("base_url") {
15                  return http.NotFoundHandler(), nil
16          }
17          return controllers.Redirector(cfg.GetString("base_url"),
            ↪  cfg.GetInt("status_code"))
18  }
19
20  func initRedirectorController(cfg *viper.Viper) error {
21          var err error
22          cfg.SetDefault("redirector.status_code", 301)
23          redirectorController, err =
            ↪  getRedirectorInstance(cfg.Sub("redirector"))
24          return err
25  }
26
27  func getRedirectorController() http.Handler {
28          return redirectorController
29  }
```

Arquivo router.go

```
1  package config
2
3  import (
4          "net/http"
5          "net/url"
6          "strings"
7
8          "github.com/fjorgemota/gurudamatricula/controllers"
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10         "github.com/fjorgemota/gurudamatricula/util/middlewares"
11
12         "github.com/justinas/alice"
13         "github.com/volatiletech/authboss/remember"
14
15         "net/http/pprof"
16
17         "github.com/spf13/viper"
18 )
19
20 var routerInstance http.Handler
21 var rootURL url.URL
22 var routes map[string]string
23
24 func getRouterInstance(cfg *viper.Viper) http.Handler {
25         router := http.NewServeMux()
26         auth := GetAuthBoss()
27         chain := alice.New(
28                 // confirm.Middleware(auth),
29                 // lock.Middleware(auth),
30                 GetGzip(),
31                 GetCors(),
32                 GetCsrf(),
33                 getDataloaderMiddleware(),
34                 remember.Middleware(auth),
35                 auth.LoadClientStateMiddleware,
36                 middlewares.ContextRequest,
37                 // authboss.Middleware2(auth, 0, 0),
38         )
39         for route, path := range routes {
```

```
40                  var handler http.Handler
41                  switch route {
42                  case "index":
43                          handler = getIndexController()
44                  case "graphql_playground":
45                          handler = getGraphQLPlaygroundController()
46                  case "graphql":
47                          handler = getGraphQLController()
48                  case "static":
49                          handler = getStaticController()
50                  case "ufsc2offers":
51                          handler = GetUFSC2OffersBotController()
52                  case "ufsc2habilitations":
53                          handler = GetUFSC2HabilitationsBotController()
54                  case "usp2offers":
55                          handler = GetUSP2OffersBotController()
56                  case "usp2habilitations":
57                          handler = GetUSP2HabilitationsBotController()
58                  case "taskqueue":
59                          handler = GetTaskQueueWorker()
60                  case "fetcher":
61                          handler = GetFetcherController()
62                  case "auth":
63                          handler =
                            ↪  http.StripPrefix(strings.TrimSuffix(path,
                            ↪  "/"), auth.Config.Core.Router)
64                  case "csrf":
65                          handler = http.HandlerFunc(controllers.CSRFToken)
66                  case "manager":
67                          handler =
                            ↪  http.StripPrefix(strings.TrimSuffix(path,
                            ↪  "/"), getManagerRouter())
68                  case "schemas":
69                          handler =
                            ↪  http.StripPrefix(strings.TrimSuffix(path,
                            ↪  "/"),
                            ↪  controllers.StaticFiles(format.GetFileSystem()))
70                  case "redirector":
71                          handler = getRedirectorController()
```

```
72                  case "pprof":
73                          handler = http.HandlerFunc(pprof.Index)
74                  }
75                  if handler == nil {
76                          panic("Unrecognized route '" + route + "'")
77                  }
78                  if len(path) > 0 && path[0] != '/' {
79                          continue
80                  }
81                  if route != "taskqueue" {
82                          handler = chain.Then(handler)
83                  }
84                  router.Handle(path, handler)
85          }
86
87          return router
88  }
89
90  func initRoutes(cfg *viper.Viper) error {
91          cfg.SetDefault("router.routes.index", "/")
92          cfg.SetDefault("router.routes.graphql_playground",
            ↪  "/api/playground/")
93          cfg.SetDefault("router.routes.graphql", "/api/graphql")
94          cfg.SetDefault("router.routes.csrf", "/api/csrf")
95          cfg.SetDefault("router.routes.static", "/static/")
96          cfg.SetDefault("router.routes.auth", "/auth/")
97          cfg.SetDefault("router.routes.fetcher", "/fetcher")
98          cfg.SetDefault("router.routes.manager", "/_internal/manager/")
99          cfg.SetDefault("router.routes.schemas", "/fetcher/schemas/")
100         cfg.SetDefault("router.routes.taskqueue",
            ↪  "/_internal/taskqueue/worker")
101         cfg.SetDefault("router.routes.ufsc2offers",
            ↪  "/_internal/publishers/ufsc2offers")
102         cfg.SetDefault("router.routes.ufsc2habilitations",
            ↪  "/_internal/publishers/ufsc2habilitations")
103         cfg.SetDefault("router.routes.usp2offers",
            ↪  "/_internal/publishers/usp2offers")
104         cfg.SetDefault("router.routes.usp2habilitations",
            ↪  "/_internal/publishers/usp2habilitations")
```

```
105         cfg.SetDefault("router.routes.pprof", "/debug/pprof/")
106         cfg.SetDefault("router.protocol", "http")
107         cfg.SetDefault("router.path", "/")
108         routes = cfg.GetStringMapString("router.routes")
109         rootURL = url.URL{
110                 Scheme: cfg.GetString("router.protocol"),
111                 Host:   cfg.GetString("router.host"),
112                 Path:   strings.TrimSuffix(cfg.GetString("router.path"),
                    ↪  "/"),
113         }
114         return nil
115 }
116
117 func initRouter(cfg *viper.Viper) error {
118         routerInstance = getRouterInstance(cfg.Sub("router"))
119         return nil
120 }
121
122 func GetRouter() http.Handler {
123         return routerInstance
124 }
125
126 func GetRoute(name string) string {
127         return routes[name]
128 }
129
130 func GetRootURLScheme() string {
131         return rootURL.Scheme
132 }
133
134 func GetRootURL() string {
135         return rootURL.String()
136 }
```

Arquivo server.go

```
1 package config
2
3 import (
4         "net/http"
```

```
5
6           "github.com/spf13/viper"
7   )
8
9   type serverTSLConfig struct {
10          certFile string
11          keyFile  string
12  }
13
14  var serverInstance *http.Server
15  var serverConfig *serverTSLConfig
16
17  func getServerInstance(cfg *viper.Viper) *http.Server {
18          return &http.Server{
19                  Addr:    cfg.GetString("server.host"),
20                  Handler: GetRouter(),
21          }
22  }
23
24  func initServer(cfg *viper.Viper) error {
25          cfg.SetDefault("server.host", cfg.GetString("router.host"))
26          serverInstance = getServerInstance(cfg)
27          serverConfig = nil
28          if cfg.IsSet("server.cert_file") && cfg.IsSet("server.key_file")
            ↪ {
29                  serverConfig = &serverTSLConfig{
30                          certFile: cfg.GetString("server.cert_file"),
31                          keyFile:  cfg.GetString("server.key_file"),
32                  }
33          }
34          return nil
35  }
36
37  func GetServer() *http.Server {
38          return serverInstance
39  }
40
41  func ListenAndServe() {
42          if serverConfig == nil {
```

```
43                 go serverInstance.ListenAndServe()
44         } else {
45                 go serverInstance.ListenAndServeTLS(serverConfig.certFile,
                   ↪  serverConfig.keyFile)
46         }
47 }
```

Arquivo static_controller.go

```
1  package config
2
3  import (
4          "net/http"
5          "os"
6
7          "github.com/fjorgemota/gurudamatricula/controllers"
8
9          "github.com/gobuffalo/packr/v2"
10         "github.com/spf13/viper"
11 )
12
13 //go:generate packr2
14
15 var defaultBox = packr.New("frontend", "./frontend")
16 var staticFileSystem http.FileSystem
17
18 func getStaticFileSystemInstance(cfg *viper.Viper) http.FileSystem {
19         var filesystem http.FileSystem = defaultBox
20         if cfg != nil {
21                 name := cfg.GetString("path")
22                 if stat, err := os.Stat(name); err == nil && stat.IsDir()
                   ↪  {
23                         filesystem = http.Dir(name)
24                 }
25         }
26         return filesystem
27 }
28
29 func initStaticFileSystem(cfg *viper.Viper) error {
30         staticFileSystem = getStaticFileSystemInstance(cfg.Sub("static"))
```

```
31          return nil
32  }
33
34  func getStaticFileSystem() http.FileSystem {
35          return staticFileSystem
36  }
37
38  func getStaticController() http.Handler {
39          return controllers.StaticFiles(getStaticFileSystem())
40  }
41
42  func getIndexController() http.Handler {
43          return controllers.Index(getStaticFileSystem())
44  }
```

Arquivo taskqueue.go

```
1   package config
2
3   import (
4           "errors"
5           "net/http"
6
7           "github.com/fjorgemota/gurudamatricula/util/taskqueue"
8           "github.com/sirupsen/logrus"
9           "github.com/spf13/viper"
10          "golang.org/x/net/context"
11  )
12
13  var taskqueueDispatcher taskqueue.TaskDispatcher
14  var taskqueueWorker http.Handler
15  var getTaskQueueCaller func(ctx context.Context) taskqueue.TaskCaller
16
17  func getTaskQueueDispatcherInstance(cfg *viper.Viper)
    ↪  (taskqueue.TaskDispatcher, error) {
18          if cfg == nil {
19                  cfg = viper.New()
20          }
21          var result taskqueue.TaskDispatcher
22          cod, err := getCoderInstance(cfg.Sub("coder"))
```

```
23          if err == nil {
24                  result = taskqueue.NewDispatcher(cod)
25          }
26          return result, err
27  }
28
29  func getTaskQueueCallerCallback(dispatcher taskqueue.TaskDispatcher, cfg
    ↪  *viper.Viper) (func(ctx context.Context) taskqueue.TaskCaller, error)
    ↪  {
30          if cfg == nil {
31                  if getTaskQueueCaller != nil {
32                          return GetTaskQueueCaller, nil
33                  }
34                  cfg = viper.New()
35          }
36          var err error
37          base := GetRootURL() + GetRoute("taskqueue")
38          switch cfg.GetString("type") {
39          case "inline":
40                  var getLogger func(ctx context.Context)
                    ↪  logrus.FieldLogger
41                  var getHTTPClient func(ctx context.Context, jar
                    ↪  http.CookieJar) *http.Client
42                  getLogger, err = getLoggerCallback(cfg.Sub("log"))
43                  getHTTPClient = getHTTPClientCallback(cfg.Sub("client"))
44                  retries := cfg.GetInt("retries")
45                  concurrency := cfg.GetInt("concurrency")
46                  clk := GetClock()
47                  return func(ctx context.Context) taskqueue.TaskCaller {
48                          return taskqueue.NewInlineCaller(dispatcher,
                            ↪  taskqueue.InlineCallerOptions{
49                                  Base:        base,
50                                  Retries:     retries,
51                                  Concurrency: concurrency,
52                                  Logger:      getLogger(ctx),
53                                  Clock:       clk,
54                                  Client:      getHTTPClient(ctx, nil),
55                          })
56                  }, err
```

```
57          case "appengine":
58                  options := taskqueue.AppEngineOptions{
59                          Base:      base,
60                          Retries:   cfg.GetInt("retries"),
61                          QueueName: cfg.GetString("queue_name"),
62                  }
63                  return func(ctx context.Context) taskqueue.TaskCaller {
64                          return taskqueue.NewAppEngineCaller(ctx,
                            ↪  dispatcher, options)
65                  }, err
66          default:
67                  err = errors.New("Unrecognized caller")
68          }
69          return func(ctx context.Context) taskqueue.TaskCaller {
70                  return nil
71          }, err
72  }
73
74  func getTaskQueueWorker(dispatcher taskqueue.TaskDispatcher, cfg
    ↪  *viper.Viper) (http.Handler, error) {
75          if cfg == nil {
76                  if taskqueueWorker != nil {
77                          return taskqueueWorker, nil
78                  }
79                  cfg = viper.New()
80          }
81          var err error
82          var worker http.Handler
83          var handler handlerDependencies
84          handler.getLogger, err = getLoggerCallback(cfg.Sub("log"))
85          if err == nil {
86                  worker = taskqueue.NewHTTPWorker(dispatcher, handler)
87          }
88          return worker, err
89  }
90
91  func initTaskQueue(cfg *viper.Viper) error {
92          cfg.SetDefault("taskqueue.caller.type", "inline")
93          subcfg := cfg.Sub("taskqueue")
```

```
94          var err error
95          taskqueueDispatcher, err =
            ↪  getTaskQueueDispatcherInstance(subcfg.Sub("dispatcher"))
96          if err == nil {
97                  getTaskQueueCaller, err =
                    ↪  getTaskQueueCallerCallback(taskqueueDispatcher,
                    ↪  subcfg.Sub("caller"))
98          }
99          if err == nil {
100                 taskqueueWorker, err =
                    ↪  getTaskQueueWorker(taskqueueDispatcher,
                    ↪  subcfg.Sub("worker"))
101         }
102         return err
103 }
104
105 func GetTaskQueueCaller(ctx context.Context) taskqueue.TaskCaller {
106         return getTaskQueueCaller(ctx)
107 }
108
109 func GetTaskQueueDispatcher() taskqueue.TaskDispatcher {
110         return taskqueueDispatcher
111 }
112
113 func GetTaskQueueWorker() http.Handler {
114         return taskqueueWorker
115 }
```

Arquivo ufsc2habilitations.go

```
1  package config
2
3  import (
4          "net/http"
5
6          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
7          "github.com/spf13/viper"
8  )
9
10 var ufsc2HabilitationsBotController http.Handler = http.NotFoundHandler()
```

```
11
12 func getUFSC2HabilitationsBotDependencies(cfg *viper.Viper)
   ↪  (handlerDependencies, error) {
13         cfg = cfg.Sub("dependencies")
14         if cfg == nil {
15                 cfg = viper.New()
16         }
17         var dependencies handlerDependencies
18         var err error
19         dependencies.getManager, err =
           ↪  getManagerCallback(cfg.Sub("manager"))
20         if err == nil {
21                 dependencies.getLogger, err =
                   ↪  getLoggerCallback(cfg.Sub("log"))
22         }
23         return dependencies, err
24 }
25
26 func getUFSC2HabilitationsBotControllerInstance(cfg *viper.Viper)
   ↪  (http.Handler, error) {
27         if !hasRequiredPublisherOptions(cfg, []string{"university_id",
           ↪  "delete_secret"}) {
28                 return http.NotFoundHandler(), nil
29         }
30         dispatcher := GetPipelineDispatcher()
31         dependencies, err := getUFSC2HabilitationsBotDependencies(cfg)
32         var result http.Handler
33         if err == nil {
34                 result, err = getPublisherController(cfg,
                   ↪  ddiffer.SourceHabilitations, dispatcher, nil,
                   ↪  dependencies)
35         }
36         return result, err
37 }
38
39 func initUFSC2HabilitationsBot(cfg *viper.Viper) error {
40         var err error
41         ufsc2HabilitationsBotController, err =
           ↪  getUFSC2HabilitationsBotControllerInstance(cfg.Sub("ufsc2.habilitations
```

```
42          return err
43  }
44
45  func GetUFSC2HabilitationsBotController() http.Handler {
46          return ufsc2HabilitationsBotController
47  }
```

Arquivo ufsc2offers.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
7           "github.com/fjorgemota/gurudamatricula/robot/ufsc2offers"
8           "github.com/spf13/viper"
9   )
10
11  var ufsc2OffersBotController http.Handler = http.NotFoundHandler()
12
13  func getUFSC2OffersBotControllerInstance(cfg *viper.Viper) (http.Handler,
    ↪  error) {
14          if !hasRequiredPublisherOptions(cfg, nil) {
15                  return http.NotFoundHandler(), nil
16          }
17          cod, err := getCoderInstance(cfg.Sub("dependencies.coder"))
18          dispatcher := GetPipelineDispatcher()
19          clk := GetClock()
20          var dependencies handlerDependencies
21          if err == nil {
22                  dependencies, err = getBotDependencies(cfg)
23          }
24          var botOptions ufsc2offers.Options
25          botOptions.BatchSize = cfg.GetInt("options.batch_size")
26          botOptions.CheckTable = cfg.GetString("options.check_table")
27          botOptions.MaxRetries = cfg.GetInt("options.max_retries")
28          botOptions.UserAgent = cfg.GetString("options.user_agent")
29          botOptions.DelayBetweenRequests =
            ↪  cfg.GetDuration("options.delay_between_requests")
```

```
30          botOptions.NumberOfPeriods =
            ↪ cfg.GetInt("options.number_of_periods")
31          botOptions.Tasks.Start = cfg.GetString("options.tasks.start")
32          botOptions.Tasks.JoinFiles =
            ↪ cfg.GetString("options.tasks.join_files")
33          botOptions.Tasks.FetchCampus =
            ↪ cfg.GetString("options.tasks.fetch_campus")
34          botOptions.Tasks.DeleteFiles =
            ↪ cfg.GetString("options.tasks.delete_files")
35          var result http.Handler
36          var bot ufsc2offers.Bot
37          if err == nil {
38                  bot, err = ufsc2offers.NewBot(dispatcher, cod, clk,
                    ↪ dependencies, botOptions)
39          }
40          if err == nil {
41                  result, err = getPublisherController(cfg,
                    ↪ ddiffer.SourceOffers, dispatcher, bot, dependencies)
42          }
43          return result, err
44  }
45
46  func initUFSC2OffersBot(cfg *viper.Viper) error {
47          var err error
48          ufsc2OffersBotController, err =
            ↪ getUFSC2OffersBotControllerInstance(cfg.Sub("ufsc2.offers"))
49          return err
50  }
51
52  func GetUFSC2OffersBotController() http.Handler {
53          return ufsc2OffersBotController
54  }
```

Arquivo usp2habilitations.go

```
1   package config
2
3   import (
4           "net/http"
5
```

```
6         "github.com/fjorgemota/gurudamatricula/robot/usp2habilitations"
7
8         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
9         "github.com/spf13/viper"
10 )
11
12 var usp2HabilitationsBotController http.Handler = http.NotFoundHandler()
13
14 func getUSP2HabilitationsBotControllerInstance(cfg *viper.Viper)
   ↪  (http.Handler, error) {
15         if !hasRequiredPublisherOptions(cfg, nil) {
16                 return http.NotFoundHandler(), nil
17         }
18         dispatcher := GetPipelineDispatcher()
19         clk := GetClock()
20         dependencies, err := getBotDependencies(cfg)
21         var botOptions usp2habilitations.PipelineOptions
22         botOptions.BatchSize = cfg.GetInt("options.batch_size")
23         botOptions.MaxRetries = cfg.GetInt("options.max_retries")
24         botOptions.Crawler.Base.DelayBetweenRequests =
           ↪  cfg.GetDuration("options.delay_between_requests")
25         botOptions.Crawler.Base.UserAgent =
           ↪  cfg.GetString("options.user_agent")
26         botOptions.Crawler.GetDisciplines = true
27         botOptions.ParallelRequests =
           ↪  cfg.GetInt("options.parallel_requests")
28         botOptions.Tasks.Start = cfg.GetString("options.tasks.start")
29         botOptions.Tasks.JoinFiles =
           ↪  cfg.GetString("options.tasks.join_files")
30         botOptions.Tasks.FetchCampi =
           ↪  cfg.GetString("options.tasks.fetch_campi")
31         botOptions.Tasks.BalanceHabilitations =
           ↪  cfg.GetString("options.tasks.balance_habilitations")
32         botOptions.Tasks.FetchHabilitations =
           ↪  cfg.GetString("options.tasks.fetch_habilitations")
33         var bot usp2habilitations.Bot
34         var result http.Handler
35         if err == nil {
```

```
36                  bot, err = usp2habilitations.NewBot(dispatcher, clk,
                    ↪  dependencies, botOptions)
37          }
38          if err == nil {
39                  result, err = getPublisherController(cfg,
                    ↪  ddiffer.SourceHabilitations, dispatcher, bot,
                    ↪  dependencies)
40          }
41          return result, err
42  }
43
44  func initUSP2HabilitationsBot(cfg *viper.Viper) error {
45          var err error
46          usp2HabilitationsBotController, err =
            ↪  getUSP2HabilitationsBotControllerInstance(cfg.Sub("usp2.habilitations")
47          return err
48  }
49
50  func GetUSP2HabilitationsBotController() http.Handler {
51          return usp2HabilitationsBotController
52  }
```

Arquivo usp2offers.go

```
1   package config
2
3   import (
4           "net/http"
5
6           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
7           "github.com/fjorgemota/gurudamatricula/robot/usp2offers"
8           "github.com/spf13/viper"
9   )
10
11  var usp2OffersBotController http.Handler = http.NotFoundHandler()
12
13  func getUSP2OffersBotControllerInstance(cfg *viper.Viper) (http.Handler,
    ↪  error) {
14          if !hasRequiredPublisherOptions(cfg, nil) {
15                  return http.NotFoundHandler(), nil
```

```
16          }
17          dispatcher := GetPipelineDispatcher()
18          clk := GetClock()
19          dependencies, err := getBotDependencies(cfg)
20          var botOptions usp2offers.Options
21          botOptions.BatchSize = cfg.GetInt("options.batch_size")
22          botOptions.MaxRetries = cfg.GetInt("options.max_retries")
23          botOptions.DelayBetweenRequests =
            ↪  cfg.GetDuration("options.delay_between_requests")
24          botOptions.ParallelRequests =
            ↪  cfg.GetInt("options.parallel_requests")
25          botOptions.UserAgent = cfg.GetString("options.user_agent")
26          botOptions.Tasks.Start = cfg.GetString("options.tasks.start")
27          botOptions.Tasks.JoinFiles =
            ↪  cfg.GetString("options.tasks.join_files")
28          botOptions.Tasks.FetchCampi =
            ↪  cfg.GetString("options.tasks.fetch_campi")
29          botOptions.Tasks.BalanceCampi =
            ↪  cfg.GetString("options.tasks.balance_campi")
30          botOptions.Tasks.FetchData =
            ↪  cfg.GetString("options.tasks.fetch_data")
31          var result http.Handler
32          var bot usp2offers.Bot
33          if err == nil {
34                  bot, err = usp2offers.NewBot(dispatcher, clk,
                    ↪  dependencies, botOptions)
35          }
36          if err == nil {
37                  result, err = getPublisherController(cfg,
                    ↪  ddiffer.SourceOffers, dispatcher, bot, dependencies)
38          }
39          return result, err
40  }
41
42  func initUSP2OffersBot(cfg *viper.Viper) error {
43          var err error
44          usp2OffersBotController, err =
            ↪  getUSP2OffersBotControllerInstance(cfg.Sub("usp2.offers"))
45          return err
```

```
46 }
47
48 func GetUSP2OffersBotController() http.Handler {
49         return usp2OffersBotController
50 }
```

B.1.3 Pasta controllers

Arquivo csrf.go

```
1  package controllers
2
3  import (
4          "io"
5          "net/http"
6
7          "github.com/gorilla/csrf"
8  )
9
10 func CSRFToken(resp http.ResponseWriter, req *http.Request) {
11         io.WriteString(resp, csrf.Token(req))
12 }
```

Arquivo fetcher.go

```
1  package controllers
2
3  import (
4          "context"
5          "errors"
6          "io"
7          "net/http"
8
9          "github.com/sirupsen/logrus"
10
11         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
12
13         "github.com/fjorgemota/gurudamatricula/robot/fetcher"
14         "github.com/fjorgemota/gurudamatricula/util/pipeline"
15 )
```

```
16
17 type FetcherDependencies interface {
18         GetPipeline(ctx context.Context) (pipeline.Pipeline, error)
19         GetLogger(ctx context.Context) logrus.FieldLogger
20         GetContext(req *http.Request) context.Context
21 }
22
23 type fetcherController struct {
24         dependencies FetcherDependencies
25         handler      fetcher.Fetcher
26 }
27
28 func (fc fetcherController) ServeHTTP(resp http.ResponseWriter, req
   ↪  *http.Request) {
29         var source ddiffer.SourceType
30         var err error
31         var pipe pipeline.Pipeline
32         var showErrorMessage bool
33         ctx := fc.dependencies.GetContext(req)
34         logger := fc.dependencies.GetLogger(ctx)
35         query := req.URL.Query()
36         sourceType := query.Get("source_type")
37         universityID := query.Get("university_id")
38         secret := query.Get("secret")
39         filename := query.Get("filename")
40         logger = logger.WithField("source_type", sourceType)
41         logger = logger.WithField("university_id", universityID)
42         logger = logger.WithField("filename", filename)
43         logger.Info("Receiving notification on fetcher")
44         switch sourceType {
45         case "habilitations":
46                 source = ddiffer.SourceHabilitations
47         case "offers":
48                 source = ddiffer.SourceOffers
49         default:
50                 err = errors.New("Invalid source type")
51                 showErrorMessage = true
52         }
53         if len(universityID) == 0 && err == nil {
```

```
54                  err = errors.New("University ID cannot be empty")
55                  showErrorMessage = true
56          }
57          if len(secret) == 0 && err == nil {
58                  err = errors.New("Secret cannot be empty")
59                  showErrorMessage = true
60          }
61          if len(filename) == 0 && err == nil {
62                  err = errors.New("File path cannot be empty")
63                  showErrorMessage = true
64          }
65          if err == nil {
66                  pipe, err = fc.dependencies.GetPipeline(ctx)
67          }
68          if err == nil {
69                  err = fc.handler.Start(ctx, pipe, source, universityID,
                    ↪  filename, secret)
70          }
71          if err != nil {
72                  msg := "Internal server error"
73                  logger.WithError(err).Errorf(msg)
74                  if showErrorMessage {
75                          msg = err.Error()
76                  }
77                  http.Error(resp, msg, http.StatusInternalServerError)
78                  return
79          }
80          logger.Info("Fetcher started!")
81          resp.WriteHeader(http.StatusOK)
82          io.WriteString(resp, "OK")
83  }
84
85  func Fetcher(dependencies FetcherDependencies, handler fetcher.Fetcher)
    ↪  http.Handler {
86          return fetcherController{
87                  dependencies: dependencies,
88                  handler:      handler,
89          }
90  }
```

Arquivo file.go

```
1  package controllers
2
3  import (
4          "context"
5          "net/http"
6          "path/filepath"
7          "strings"
8          "time"
9
10         "github.com/fjorgemota/gurudamatricula/util/file"
11 )
12
13 type FileHandler interface {
14         GetManager(ctx context.Context) (file.Manager, error)
15         GetContext(req *http.Request) context.Context
16 }
17
18 type fileController struct {
19         handler FileHandler
20 }
21
22 func (fc fileController) ServeHTTP(resp http.ResponseWriter, req
   ↪  *http.Request) {
23         path := req.URL.Path
24         if !strings.HasPrefix(path, "/") {
25                 path = "/" + path
26         }
27         path = filepath.Clean(path)
28         var reader file.ReaderSeekerCloser
29         ctx := fc.handler.GetContext(req)
30         manager, err := fc.handler.GetManager(ctx)
31         if err == nil {
32                 reader, err = manager.Reader(path)
33         }
34         if err == nil {
35                 http.ServeContent(resp, req, path, time.Time{}, reader)
36         }
```

```
37          if err == nil {
38                  err = reader.Close()
39          }
40          if err != nil {
41                  if file.IsNotExistError(err) {
42                          http.Error(resp, "Not found",
                            ↪  http.StatusNotFound)
43                  } else {
44                          http.Error(resp, "Internal Server Error",
                            ↪  http.StatusInternalServerError)
45                  }
46          }
47  }
48
49  func FileController(handler FileHandler) http.Handler {
50          return fileController{
51                  handler: handler,
52          }
53  }
```

Arquivo graphql.go

```
1   package controllers
2
3   import (
4           "net/http"
5
6           "github.com/99designs/gqlgen/handler"
7           "github.com/fjorgemota/gurudamatricula/graphql"
8           "github.com/fjorgemota/gurudamatricula/graphql/resolvers"
9   )
10
11  type GraphQLPlaygroundResolver interface {
12          GetRoute(name string) string
13  }
14
15  func GraphQLController(dependencies resolvers.Handler) http.Handler {
16          resolver := resolvers.Resolver{
17                  Handler: dependencies,
18          }
```

```
19          schema := graphql.NewExecutableSchema(graphql.Config{Resolvers:
            ↪  resolver})
20          return handler.GraphQL(schema)
21  }
22
23  func GraphQLPlaygroundController(dependencies GraphQLPlaygroundResolver)
    ↪  http.Handler {
24          return handler.Playground("Guru da Matricula - GraphQL
            ↪  playground", dependencies.GetRoute("graphql"))
25  }
```

Arquivo publisher.go

```
1   package controllers
2
3   import (
4           "context"
5           "errors"
6           "fmt"
7           "net/http"
8           "path/filepath"
9           "strings"
10
11          "github.com/sirupsen/logrus"
12
13          "github.com/fjorgemota/gurudamatricula/util/pipeline"
14
15          "github.com/fjorgemota/gurudamatricula/robot/publisher"
16  )
17
18  type PublisherHandler interface {
19          GetContext(req *http.Request) context.Context
20          GetLogger(ctx context.Context) logrus.FieldLogger
21          GetPipeline(ctx context.Context) (pipeline.Pipeline, error)
22  }
23
24  type PublisherSecrets struct {
25          StartSecret  string
26          DeleteSecret string
27  }
```

```
28
29 type publisherController struct {
30         handler PublisherHandler
31         pub     publisher.Publisher
32         secrets PublisherSecrets
33 }
34
35 func (pc publisherController) ServeHTTP(resp http.ResponseWriter, req
   ↪  *http.Request) {
36         ctx := pc.handler.GetContext(req)
37         logger := pc.handler.GetLogger(ctx)
38         pipe, err := pc.handler.GetPipeline(ctx)
39         if err == nil {
40                 query := req.URL.Query()
41                 secret := query.Get("secret")
42                 if secret == "" {
43                         http.Error(resp, "Secret required",
                        ↪  http.StatusForbidden)
44                         return
45                 }
46                 switch secret {
47                 case pc.secrets.StartSecret:
48                         err = pc.pub.Start(pipe)
49                 case pc.secrets.DeleteSecret:
50                         for _, filename := range query["filenames"] {
51                                 filename = filepath.Clean(filename)
52                                 if err == nil &&
                                ↪  strings.Contains(filename, "/") {
53                                         err = errors.New("Invalid path")
54                                 }
55                                 if err == nil {
56                                         err = pc.pub.DeleteFile(pipe,
                                        ↪  filename)
57                                 }
58                         }
59                 default:
60                         http.Error(resp, "Invalid Secret",
                        ↪  http.StatusForbidden)
61                         return
```

```
62                  }
63          }
64          status := http.StatusInternalServerError
65          msg := "Internal error"
66          if err == nil {
67                  status = (http.StatusOK)
68                  msg = "OK"
69          } else {
70                  logger.WithError(err).Errorf("Error while processing
                    ↪  request to publisher")
71          }
72          resp.Header().Set("Content-Type", "text/plain; charset=utf-8")
73          resp.WriteHeader(status)
74          fmt.Fprintln(resp, msg)
75  }
76
77  func Publisher(handler PublisherHandler, pub publisher.Publisher, secrets
    ↪  PublisherSecrets) http.Handler {
78          return publisherController{
79                  handler: handler,
80                  pub:     pub,
81                  secrets: secrets,
82          }
83  }
```

Arquivo redirector.go

```
1   package controllers
2
3   import (
4           "net/http"
5           "net/url"
6   )
7
8   type redirectController struct {
9           base   *url.URL
10          status int
11  }
12
```

```
13 func (rc redirectController) ServeHTTP(resp http.ResponseWriter, req
   ↪  *http.Request) {
14         redirTo := rc.base.ResolveReference(req.URL)
15         http.Redirect(resp, req, redirTo.String(), rc.status)
16 }
17
18 func Redirector(baseURL string, status int) (http.Handler, error) {
19         var result http.Handler
20         base, err := url.Parse(baseURL)
21         if err == nil {
22                 result = redirectController{
23                         base:   base,
24                         status: status,
25                 }
26         }
27         return result, err
28 }
```

Arquivo static.go

```
1  package controllers
2
3  import (
4          "net/http"
5          "os"
6          "path"
7  )
8
9  func StaticFiles(fs http.FileSystem) http.Handler {
10         return http.FileServer(fs)
11 }
12
13 func toHTTPError(err error) (string, int) {
14         if os.IsNotExist(err) {
15                 return "404 page not found", http.StatusNotFound
16         }
17         if os.IsPermission(err) {
18                 return "403 Forbidden", http.StatusForbidden
19         }
```

```
20          return "500 Internal Server Error",
            ↪  http.StatusInternalServerError
21  }
22
23  type indexController struct {
24          filesystem http.FileSystem
25  }
26
27  func (ic indexController) ServeHTTP(resp http.ResponseWriter, req
    ↪  *http.Request) {
28          var stat os.FileInfo
29          var reader http.File
30          requestURI := path.Clean(req.RequestURI)
31          filename := path.Base(requestURI)
32          reader, err := ic.filesystem.Open(requestURI)
33          if err == nil {
34                  stat, err = reader.Stat()
35          }
36          if err != nil || stat.IsDir() {
37                  filename = "index.html"
38                  reader, err = ic.filesystem.Open("index.html")
39          }
40          if err == nil {
41                  stat, err = reader.Stat()
42          }
43          if err == nil {
44                  http.ServeContent(resp, req, filename, stat.ModTime(),
                    ↪  reader)
45          }
46          if err == nil {
47                  err = reader.Close()
48          }
49          if err != nil {
50                  msg, code := toHTTPError(err)
51                  http.Error(resp, msg, code)
52          }
53  }
54
55  func Index(fs http.FileSystem) http.Handler {
```

```
56          return indexController{
57                  filesystem: fs,
58          }
59  }
```

B.1.4 Pasta graphql

Arquivo base.go

```
1  package graphql
2
3  //go:generate go run ../tools/gqlgen/main.go -v
```

Arquivo base_models.go

```
1  package graphql
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5  )
6
7  type DisciplineSummary struct {
8          ID           string            `json:"id"`
9          UniversityID string            `json:"university_id"`
10         Course       models.ForeignKey `json:"course"`
11         Habilitation models.ForeignKey `json:"habilitation"`
12         Step         models.ForeignKey `json:"step"`
13         GenericID    string            `json:"generic_id"`
14         Version      string            `json:"version"`
15         Code         string            `json:"code"`
16         Name         string            `json:"name"`
17         Description  string            `json:"description"`
18         Type         string            `json:"type"`
19 }
20
21 type Discipline struct {
22         ID                 string                      `json:"id"`
23         UniversityID       string
           ↪  `json:"university_id"`
24         Course             models.ForeignKey           `json:"course"`
```

```
25          Habilitation       models.ForeignKey
            ↪ `json:"habilitation"`
26          Step               models.ForeignKey          `json:"step"`
27          GenericID          string
            ↪ `json:"generic_id"`
28          Version            string                     `json:"version"`
29          Code               string                     `json:"code"`
30          Name               string                     `json:"name"`
31          Description        string
            ↪ `json:"description"`
32          Type               string                     `json:"type"`
33          Workload           []models.DisciplineWorkload `json:"workload"`
34          Related            []DisciplineRelated        `json:"related"`
35          RelatedDisciplines []DisciplineSummary
            ↪ `json:"related_disciplines"`
36          Tables             []models.Table             `json:"tables"`
37          TableColumns       []models.TableColumn
            ↪ `json:"tables_columns"`
38          TableRows          []models.TableRow
            ↪ `json:"tables_rows"`
39  }
40
41  type Step struct {
42          HabilitationID string              `json:"-"`
43          ID             string              `json:"id"`
44          Name           string              `json:"name"`
45          Disciplines    []models.ForeignKey `json:"disciplines"`
46  }
```

Arquivo gqlgen.yml

```
1  schema: schema.graphql
2  exec:
3    filename: server.generated.go
4    package: graphql
5  model:
6    filename: models.generated.go
7    package: graphql
8  models:
9    Campus:
```

```
10      model: github.com/fjorgemota/gurudamatricula/models.Campus
11    CourseOffer:
12      model:
        ↪ github.com/fjorgemota/gurudamatricula/models.DisciplineOfferTeamCourse
13    Course:
14      model: github.com/fjorgemota/gurudamatricula/models.Course
15    Discipline:
16      model: github.com/fjorgemota/gurudamatricula/models.Discipline
17    DisciplineOffer:
18      model: github.com/fjorgemota/gurudamatricula/models.DisciplineOffer
19      fields:
20        teams:
21          resolver: true
22    ForeignKeyInput:
23      model: github.com/fjorgemota/gurudamatricula/models.ForeignKey
24    Habilitation:
25      model: github.com/fjorgemota/gurudamatricula/models.Habilitation
26    HourMinute:
27      model:
        ↪ github.com/fjorgemota/gurudamatricula/models.DisciplineOfferTeamScheduleHou
28    Period:
29      model: github.com/fjorgemota/gurudamatricula/models.Period
30    Plan:
31      model: github.com/fjorgemota/gurudamatricula/models.Plan
32    PlanVersion:
33      model: github.com/fjorgemota/gurudamatricula/models.PlanVersion
34    PlanVacancyInput:
35      model:
        ↪ github.com/fjorgemota/gurudamatricula/models.DisciplineOfferTeamVacancy
36    PlanDisciplineData:
37      model: github.com/fjorgemota/gurudamatricula/graphql.Discipline
38    PlanDisciplineSummary:
39      model:
        ↪ github.com/fjorgemota/gurudamatricula/graphql.DisciplineSummary
40    PlanDisciplineOfferData:
41      model: github.com/fjorgemota/gurudamatricula/models.DisciplineOffer
42    Room:
43      model: github.com/fjorgemota/gurudamatricula/models.Room
44    Step:
```

```
45      model: github.com/fjorgemota/gurudamatricula/graphql.Step
46      fields:
47        disciplines:
48          resolver: true
49    TableColumn:
50      model: github.com/fjorgemota/gurudamatricula/models.TableColumn
51    TableRow:
52      model: github.com/fjorgemota/gurudamatricula/models.TableRow
53    Teacher:
54      model: github.com/fjorgemota/gurudamatricula/models.Teacher
55    Team:
56      fields:
57        campus:
58          resolver: true
59        period:
60          resolver: true
61        disciplinesInfo:
62          resolver: true
63        university:
64          resolver: true
65    University:
66      model: github.com/fjorgemota/gurudamatricula/models.University
67    UniversityData:
68      model: github.com/fjorgemota/gurudamatricula/models.UniversityData
69      fields:
70        secret:
71          resolver: true
72        baseUrl:
73          resolver: true
74        deleteUrl:
75          resolver: true
76    User:
77      model: github.com/fjorgemota/gurudamatricula/models.User
78    UserDiscipline:
79      model: github.com/fjorgemota/gurudamatricula/models.UserDiscipline
80    UserDisciplineInput:
81      model: github.com/fjorgemota/gurudamatricula/models.UserDiscipline
82    Vacancy:
```

```
83      model:
        ↪  github.com/fjorgemota/gurudamatricula/models.DisciplineOfferTeamVacancy
```

Arquivo schema.graphql

```
1  schema {
2    query: Query
3    mutation: Mutation
4  }
5
6  ################### Mutation/Inputs ###################
7  input PlanHourMinuteInput {
8    hour: Int!
9    minute: Int!
10 }
11
12 input PlanRoomInput {
13   id: ID!
14   genericID: String!
15   name: String!
16   olcode: String
17   description: String
18 }
19
20 input PlanScheduleInput {
21   start: PlanHourMinuteInput!
22   end: PlanHourMinuteInput!
23   dayOfWeek: Int!
24   room: PlanRoomInput
25 }
26
27 input PlanTeacherInput {
28   id: ID!
29   genericID: String!
30   name: String!
31   url: String
32 }
33
34 input PlanCampusInput {
35   id: ID!
```

```
36   name: String!
37 }
38
39 input PlanVacancyInput {
40   offered: Int!
41   filled: Int!
42 }
43
44 input PlanTableInput {
45   id: ID!
46   title: String!
47   description: String
48   columns: [PlanTableColumnInput!]!
49   rows: [PlanTableRowInput!]!
50 }
51
52 input PlanTableColumnInput {
53   id: ID!
54   name: String!
55 }
56
57 input PlanTableRowInput {
58   row: Int!
59   column: Int!
60   value: String!
61 }
62
63 input PlanCourseOfferInput {
64   id: ID!
65   name: String!
66   exclusivity: Exclusivity! = Unknown
67 }
68
69 input PlanTeamInput {
70   id: ID!
71   code: String!
72   version: String!
73   schedules: [PlanScheduleInput!]!
74   teachers: [PlanTeacherInput!]!
```

```
  vacancies: PlanVacancyInput
  course: PlanCourseOfferInput
  tables: [PlanTableInput!]!
}

input PlanDisciplineOfferInput {
  id: ID!
  genericID: ID
  campus: ForeignKeyInput
  version: String!
  custom: Boolean!
  code: String!
  name: String!
  teams: [PlanTeamInput!]!
}

input PlanDisciplineRelatedItemInput {
  type: String!
  value: String!
  description: String
}

input PlanDisciplineRelatedInput {
  name: String!
  type: String!
  items: [PlanDisciplineRelatedItemInput!]!
}

input PlanDisciplineRelatedDisciplineSummary {
  id: ID!
  genericID: ID!
  course: ForeignKeyInput!
  habilitation: ForeignKeyInput!
  step: ForeignKeyInput!
  version: String!
  code: String!
  name: String!
  type: String!
  description: String!
```

```
114 }
115
116 input PlanDisciplineInput {
117   id: ID!
118   genericID: ID!
119   course: ForeignKeyInput!
120   habilitation: ForeignKeyInput!
121   step: ForeignKeyInput!
122   version: String!
123   code: String!
124   name: String!
125   type: String!
126   description: String!
127   related: [PlanDisciplineRelatedInput!]!
128   relatedDisciplines: [PlanDisciplineRelatedDisciplineSummary!]!
129   tables: [PlanTableInput!]!
130 }
131
132 input PlanTeamSelectionInput {
133   offerID: ID!
134   offerVersion: String!
135   teamID: ID!
136   color: String!
137   enabled: Boolean!
138   selected: Boolean!
139 }
140
141 input PlanDisciplineSelectionInput {
142   disciplineID: ID!
143   disciplineVersion: String!
144   enabled: Boolean!
145 }
146
147 input PlanPossibilityInput {
148   name: String!
149   selectionTeams: [PlanTeamSelectionInput!]!
150   selectionDisciplines: [PlanDisciplineSelectionInput!]!
151 }
152
```

```
153 input ForeignKeyInput {
154   id: ID!
155   name: String!
156 }
157
158 input PlanInput {
159   id: ID
160   name: String!
161   public: Boolean!
162   universityID: ID!
163   periodID: ID!
164   disciplines: [PlanDisciplineInput!]!
165   offers: [PlanDisciplineOfferInput!]!
166   possibilities: [PlanPossibilityInput!]!
167   selectedPossibility: Int!
168 }
169
170 input UserPasswordInput {
171   oldPassword: String!
172   password: String!
173   confirmPassword: String!
174 }
175
176 input UserDisciplineInput {
177   id: ID!
178   genericID: ID!
179   code: String!
180   name: String!
181 }
182
183 input UserInput {
184   name: String!
185   password: UserPasswordInput
186   university: ForeignKeyInput
187   courses: [ForeignKeyInput!]
188   habilitations: [ForeignKeyInput!]
189   completedDisciplines: [UserDisciplineInput!]!
190 }
191
```

```
192 input UniversityDataInput {
193   baseUrl: String!
194   deleteUrl: String!
195   aboutUrl: String!
196   generateSecret: Boolean!
197 }
198
199 input UniversityInput {
200   id: ID
201   acronym: String!
202   name: String!
203   public: Boolean!
204   habilitations: UniversityDataInput!
205   offers: UniversityDataInput!
206 }
207
208 type Mutation {
209   setUniversity(university: UniversityInput!): University!
210   setPlan(plan: PlanInput!): Plan
211   deletePlan(id: String!): Boolean!
212   setUser(user: UserInput!): User!
213   deleteUser(password: String!): Boolean!
214 }
215
216 ################### Query/Types ###################
217
218 ################ Search ################
219
220 enum SearchOperator {
221   All
222   And
223   Or
224   Not
225   Terminal
226   FieldTerminal
227 }
228
229 input SearchExpression {
230   type: SearchOperator!
```

```
231   expressions: [SearchExpression!]
232   attribute: String
233   value: String
234 }
235
236 enum SearchFilter {
237   NONE
238   PREREQUISITES
239   COURSES
240   ALL
241 }
242
243 input SearchOptions {
244   version: String
245   before: Int
246   after: Int
247   limit: Int
248   filter: SearchFilter = NONE
249 }
250
251 type SearchQueryInfo {
252   version: String!
253   last: Int!
254   first: Int!
255   count: Int!
256 }
257
258 type SearchPageInfo {
259   after: Int!
260   before: Int!
261 }
262
263 type SearchResultInfo {
264   query: SearchQueryInfo!
265   page: SearchPageInfo!
266 }
267
268 interface SearchResult {
269   info: SearchResultInfo!
```

```
}

################ Plans ################

type PlanTeamSelection {
  offerID: ID!
  offerVersion: String!
  teamID: ID!
  color: String!
  selected: Boolean!
  enabled: Boolean!
}

type PlanDisciplineSelection {
  disciplineID: ID!
  disciplineVersion: String!
  enabled: Boolean!
}

type PlanPossibility {
  name: String!
  disciplines: [PlanDisciplineSelection!]!
  teams: [PlanTeamSelection!]!
}

scalar Time

type PlanDisciplineOfferData {
  id: ID!
  genericID: ID!
  university: University!
  campus: Campus!
  period: Period!
  version: String!
  code: String!
  name: String!
  teams: [Team!]!
  team(id: ID!): Team
}
```

```
309
310 type PlanDisciplineOffer {
311   custom: Boolean!
312   deleted: Boolean!
313   updated: DisciplineOffer
314   offer: PlanDisciplineOfferData!
315 }
316
317
318 type PlanDisciplineSummary {
319   id: ID!
320   genericID: ID!
321   course: Course!
322   habilitation: Habilitation!
323   step: Step!
324   version: String!
325   code: String!
326   name: String!
327   type: String!
328   description: String!
329 }
330
331 type PlanDisciplineData {
332   id: ID!
333   version: String!
334   university: University!
335   course: Course!
336   habilitation: Habilitation!
337   step: Step!
338   genericID: ID!
339   code: String!
340   name: String!
341   description: String!
342   type: String!
343   tables: [Table!]!
344   related: [DisciplineRelated!]!
345   relatedDisciplines: [PlanDisciplineSummary!]!
346 }
347
```

```
348 type PlanDiscipline {
349   deleted: Boolean!
350   updated: Discipline
351   discipline: PlanDisciplineData!
352 }
353
354 type PlanSelectedSchedule {
355   offerID: ID!
356   title: String!
357   color: String!
358   start: HourMinute!
359   end: HourMinute!
360   dayOfWeek: Int!
361 }
362
363 type PlanVersion {
364   id: ID!
365   savedAt: Time!
366   selectedPossibility: Int!
367   disciplines: [PlanDiscipline!]!
368   disciplineOffers: [PlanDisciplineOffer!]!
369   possibilities: [PlanPossibility!]!
370   totalPossibilities: Int!
371   selectedSchedules: [PlanSelectedSchedule!]!
372 }
373
374 type Plan {
375   id: ID!
376   name: String!
377   lastModified: Time!
378   createdAt: Time!
379   user: User!
380   public: Boolean!
381   university: University!
382   period: Period!
383   versions(start: Int, end: Int): [PlanVersion!]!
384   versionByIndex(index: Int!): PlanVersion
385   versionByID(id: ID!): PlanVersion
386 }
```

```
387
388 ################ User ################
389
390 type UserDiscipline {
391   id: ID!
392   genericID: ID!
393   code: String!
394   name: String!
395 }
396
397 type User {
398   id: String!
399   name: String!
400   email: String!
401   oauth2_provider: String!
402   plans: [Plan!]!
403   university: University
404   courses: [Course!]
405   habilitations: [Habilitation!]
406   completedDisciplines: [UserDiscipline!]!
407 }
408
409 ################ Tables ################
410
411 type TableColumn {
412   id: ID!
413   name: String!
414 }
415
416 type TableRow {
417   row: Int!
418   column: Int!
419   value: String!
420 }
421
422 type Table {
423   id: ID!
424   title: String!
425   description: String
```

```
426   columns: [TableColumn!]!
427   rows: [TableRow!]!
428 }
429
430 ################ Offers ################
431
432 enum Exclusivity {
433   Exclusive
434   NotExclusive
435   Unknown
436 }
437
438 type CourseOffer {
439   id: ID!
440   name: String!
441   exclusivity: Exclusivity!
442 }
443
444 type Vacancy {
445   offered: Int!
446   filled: Int!
447 }
448
449 type Teacher {
450   id: ID!
451   genericID: String!
452   university: University!
453   period: Period!
454   name: String!
455   url: String
456   searchTeams(
457     periodID: ID
458     query: SearchExpression
459     options: SearchOptions
460   ): TeamSearchResult!
461 }
462
463 type TeacherSearchResult implements SearchResult {
464   info: SearchResultInfo!
```

```
465   results: [Teacher!]!
466 }
467
468 type HourMinute {
469   hour: Int!
470   minute: Int!
471 }
472
473 type RoomSearchResult implements SearchResult {
474   info: SearchResultInfo!
475   results: [Room!]!
476 }
477
478 type Room {
479   id: ID!
480   genericID: String!
481   university: University!
482   campus: Campus!
483   period: Period!
484   name: String!
485   olcode: String
486   description: String
487   searchTeams(
488     periodID: ID
489     query: SearchExpression
490     options: SearchOptions
491   ): TeamSearchResult!
492 }
493
494 type Schedule {
495   start: HourMinute!
496   end: HourMinute!
497   dayOfWeek: Int!
498   room: Room
499 }
500
501 type DisciplineOfferSearchResult implements SearchResult {
502   info: SearchResultInfo!
503   results: [DisciplineOffer!]!
```

```
504 }
505
506 type DisciplineOffer {
507   id: ID!
508   genericID: ID!
509   university: University!
510   campus: Campus!
511   period: Period!
512   version: String!
513   code: String!
514   name: String!
515   teams: [Team!]!
516   team(id: ID!): Team
517   searchTeams(
518     query: SearchExpression
519     options: SearchOptions
520   ): TeamSearchResult!
521 }
522
523 type TeamSearchResult implements SearchResult {
524   info: SearchResultInfo!
525   results: [Team!]!
526 }
527
528 type Team {
529   id: ID!
530   code: String!
531   version: String!
532   schedules: [Schedule!]!
533   teachers: [Teacher!]!
534   vacancies: Vacancy
535   university: University!
536   campus: Campus!
537   period: Period!
538   discipline: DisciplineOffer!
539   course: CourseOffer
540   tables: [Table!]!
541 }
542
```

```
543 type Campus {
544   id: ID!
545   university: University!
546   version: String!
547   period: Period!
548   name: String!
549   disciplineOffers: [DisciplineOffer!]!
550   searchDisciplineOffers(
551     query: SearchExpression
552     options: SearchOptions
553   ): DisciplineOfferSearchResult!
554   searchTeams(
555     query: SearchExpression
556     options: SearchOptions
557   ): TeamSearchResult!
558 }
559
560 type Period {
561   id: ID!
562   university: University!
563   version: String!
564   name: String!
565   campi: [Campus!]!
566 }
567
568 ################ Semantic ################
569
570 enum DisciplineRelatedItemType {
571   DISCIPLINE_ID,
572   DISCIPLINE_GENERIC_ID,
573   TEXT
574 }
575
576 type DisciplineRelatedItem {
577   type: DisciplineRelatedItemType!
578   value: String!
579   description: String
580 }
581
```

```
582 enum DisciplineRelatedType {
583   PREREQUISITE,
584   SET,
585   EQUIVALENT
586 }
587
588 type DisciplineRelated {
589   name: String!
590   type: DisciplineRelatedType!
591   items: [DisciplineRelatedItem!]!
592 }
593
594 type DisciplineSearchResult implements SearchResult {
595   info: SearchResultInfo!
596   results: [Discipline!]!
597 }
598
599 type Discipline {
600   id: ID!
601   version: String!
602   university: University!
603   course: Course!
604   habilitation: Habilitation!
605   step: Step!
606   genericID: ID!
607   code: String!
608   name: String!
609   description: String!
610   type: String!
611   tables: [Table!]!
612   searchTeams(
613     periodID: ID!
614     query: SearchExpression
615     options: SearchOptions
616   ): TeamSearchResult!
617   searchDisciplineOffers(
618     periodID: ID!
619     query: SearchExpression
620     options: SearchOptions
```

```
621   ): DisciplineOfferSearchResult!
622   related: [DisciplineRelated!]!
623   relatedDisciplines: [Discipline!]!
624 }
625
626 type HabilitationLimitValue {
627   id: Int!
628   key: String!
629   value: Float!
630 }
631
632 type HabilitationLimit {
633   id: ID!
634   name: String!
635   context: String!
636   unit: String!
637   values: [HabilitationLimitValue!]!
638 }
639
640 type Step {
641   id: ID!
642   name: String!
643   disciplines: [Discipline!]!
644 }
645
646 type Habilitation {
647   id: ID!
648   university: University!
649   course: Course!
650   version: String!
651   name: String!
652   steps: [Step!]!
653   tables: [Table!]!
654   limits: [HabilitationLimit!]!
655 }
656
657 type Course {
658   id: ID!
659   university: University!
```

```
660   name: String!
661   version: String!
662   courseVersion: String!
663   url: String
664   habilitations: [Habilitation!]!
665 }
666
667 type UniversityMutex {
668   lastUpdated: Time!
669 }
670
671 type UniversityData {
672   baseUrl: String!
673   deleteUrl: String!
674   aboutUrl: String!
675   lastUpdated: Time
676   queueSize: Int!
677   secret: String!
678 }
679
680 type University {
681   id: ID!
682   acronym: String!
683   name: String!
684   public: Boolean!
685   periods: [Period!]!
686   courses: [Course!]!
687   offers: UniversityData!
688   habilitations: UniversityData!
689 }
690
691 ################### Offline System ###################
692
693 input OfflineIDVersion {
694   ID: String!
695   Version: String!
696 }
697
698 input OfflineRequest {
```

```
699   universityID: String!
700   periodID: String!
701   courseID: String!
702   habilitations: [OfflineIDVersion!]!
703   disciplineOffers: [OfflineIDVersion!]!
704   disciplines: [OfflineIDVersion!]!
705 }
706
707 type OfflineResponse {
708   disciplineOffers: [DisciplineOffer!]!
709   disciplineOffersToDelete: [String!]!
710   disciplines: [Discipline!]!
711   disciplinesToDelete: [String!]!
712   habilitations: [Habilitation!]!
713   habilitationsToDelete: [String!]!
714 }
715
716 type Query {
717   university(id: ID!): University
718   universities(onlyOwned: Boolean = false): [University!]!
719   course(id: ID!): Course
720   coursesByUniversity(universityID: ID!): [Course!]!
721   coursesByIDs(ids: [ID!]!): [Course!]!
722   period(id: ID!): Period
723   periodsByUniversity(universityID: ID!): [Period!]!
724   periodsByIDs(ids: [ID!]!): [Period!]!
725   campus(id: ID!): Campus
726   campiByPeriod(periodID: ID!): [Campus!]!
727   campiByIDs(ids: [ID!]!): [Campus!]!
728   habilitation(id: ID!): Habilitation
729   habilitationsByCourse(courseID: ID!): [Habilitation!]!
730   habilitationsByCourses(coursesIDs: [ID!]!): [Habilitation!]!
731   habilitationsByIDs(ids: [ID!]!): [Habilitation!]!
732   disciplineOffer(id: ID!): DisciplineOffer
733   disciplineOffersByCampus(campusID: ID!): [DisciplineOffer!]!
734   disciplineOffersByIDs(ids: [ID!]!): [DisciplineOffer!]!
735   discipline(id: ID!): Discipline
736   disciplinesByHabilitationAndStep(habilitationID: ID!, stepID: ID!):
      ↪ [Discipline!]!
```

```
737   disciplinesByIDs(ids: [ID!]!): [Discipline!]!
738   team(disciplineID: ID!, teamID: ID!): Team
739   teams(disciplineID: ID!): [Team!]!
740   loggedUser: User
741   getOfflineUpdates(request: [OfflineRequest!]!): OfflineResponse!
742   searchTeachers(
743     periodID: ID!
744     query: SearchExpression
745     options: SearchOptions
746   ): TeacherSearchResult!
747   searchDisciplines(
748     universityID: ID!
749     query: SearchExpression
750     options: SearchOptions
751   ): DisciplineSearchResult!
752   searchDisciplineOffers(
753     periodID: ID!
754     query: SearchExpression
755     options: SearchOptions
756   ): DisciplineOfferSearchResult!
757   searchTeams(
758     periodID: ID!
759     query: SearchExpression
760     options: SearchOptions
761   ): TeamSearchResult!
762   plan(id: ID!): Plan
763   plans: [Plan!]!
764 }
```

Arquivo team.go

```
1  package graphql
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5  )
6
7  func ToTeam(row *models.DisciplineOffer, teamIndex int) Team {
8          rowTeam := row.Teams[teamIndex]
9          team := Team{
```

```
10                  ID:         rowTeam.ID,
11                  Code:       rowTeam.Code,
12                  Course:     &rowTeam.Course,
13                  Vacancies:  &rowTeam.Vacancies,
14                  Version:    row.Version,
15                  Discipline: row,
16                  Campus: &models.Campus{
17                          ID:   row.Campus.ID,
18                          Name: row.Campus.Name,
19                  },
20                  Period: &models.Period{
21                          ID:   row.Period.ID,
22                          Name: row.Period.Name,
23                  },
24                  University: &models.University{
25                          ID: row.UniversityID,
26                  },
27          }
28          for _, teamSchedule := range row.TeamSchedules {
29                  if teamSchedule.TeamID == teamIndex {
30                          schedule :=
                            ↪  row.Schedules[teamSchedule.ScheduleID]
31                          var room *models.Room
32                          if teamSchedule.RoomID >= 0 {
33                                  teamRow := row.Rooms[teamSchedule.RoomID]
34                                  room = &models.Room{
35                                          ID:           teamRow.ID,
36                                          Name:         teamRow.Name,
37                                          OLCode:       teamRow.OLCode,
38                                          Description:
                                            ↪  teamRow.Description,
39                                          Period:       row.Period,
40                                          Campus:       row.Campus,
41                                          UniversityID: row.UniversityID,
42                                  }
43                          }
44                          team.Schedules = append(team.Schedules,
                            ↪  &Schedule{
45                                  DayOfWeek: int(schedule.DayOfWeek),
```

```
46                              Start:      &schedule.Start,
47                              End:        &schedule.End,
48                              Room:       room,
49                      })
50              }
51      }
52      for _, teamTeacher := range row.TeamTeachers {
53              if teamTeacher.TeamID == teamIndex {
54                      teacher := row.Teachers[teamTeacher.TeacherID]
55                      team.Teachers = append(team.Teachers,
                        ↪  &models.Teacher{
56                              ID:           teacher.ID,
57                              Name:         teacher.Name,
58                              URL:          teacher.URL,
59                              Version:      row.Version,
60                              Period:       row.Period,
61                              UniversityID: row.UniversityID,
62                      })
63              }
64      }
65      for tableID, table := range row.TeamTables {
66              if table.TeamID == teamIndex {
67                      tableResult := Table{
68                              ID:          table.ID,
69                              Title:       table.Title,
70                              Description: &table.Description,
71                      }
72                      for _, tableColumn := range row.TableColumns {
73                              if tableColumn.TableID == tableID {
74                                      column := tableColumn
75                                      tableResult.Columns =
                                        ↪  append(tableResult.Columns,
                                        ↪  &column)
76                              }
77                      }
78                      for _, tableRow := range row.TableRows {
79                              if
                                ↪  row.TableColumns[tableRow.Column].TableID
                                ↪  == tableID {
```

```
80                                              row := tableRow
81                                              tableResult.Rows =
                                                ↪  append(tableResult.Rows,
                                                ↪  &row)
82                                      }
83                              }
84                              team.Tables = append(team.Tables, &tableResult)
85                      }
86              }
87              return team
88      }
```

B.1.5 Pasta models

Arquivo base.go

```
1   package models
2
3   import "encoding/gob"
4
5   //go:generate codecgen -o $GOPACKAGE.generated.go campus.go course.go
    ↪  discipline.go discipline_offer.go habilitation.go period.go plan.go
    ↪  room.go teacher.go team.go table.go university.go user.go
6
7   func init() {
8           gob.Register(Campus{})
9           gob.Register(Course{})
10          gob.Register(Discipline{})
11          gob.Register(DisciplineOffer{})
12          gob.Register(Habilitation{})
13          gob.Register(Period{})
14          gob.Register(Plan{})
15          gob.Register(Room{})
16          gob.Register(Teacher{})
17          gob.Register(Team{})
18          gob.Register(University{})
19          gob.Register(User{})
20  }
```

Arquivo campus.go

```
1  package models
2
3  // TableCampus constains the name of the table for campus
4  const TableCampus = "campus"
5
6  // Campus is a struct that represents a Campus of a University
7  type Campus struct {
8          ID               string       `datastore:"id,noindex" json:"id"`
9          UniversityID     string       `datastore:"university_id,noindex"
           ↪  json:"university_id"`
10         Period           ForeignKey   `datastore:"period,noindex"
           ↪  json:"period"`
11         Version          string       `datastore:"version,noindex"
           ↪  json:"version"`
12         Name             string       `datastore:"name,noindex"
           ↪  json:"name"`
13         OLCode           string       `datastore:"olcode,noindex"
           ↪  json:"olcode"`
14         DisciplineOffers []ForeignKey
           ↪  `datastore:"discipline_offers,noindex"
           ↪  json:"discipline_offers"`
15 }
```

Arquivo course.go

```
1  package models
2
3  // TableCouse constains the name of the table for campus
4  const TableCourse = "course"
5
6  // Course represents a course in a university
7  type Course struct {
8          ID            string       `datastore:"id,noindex" json:"id"`
9          UniversityID  string       `datastore:"university_id,noindex"
           ↪  json:"university_id"`
10         Version       string       `datastore:"version,noindex"
           ↪  json:"version"`
11         Name          string       `datastore:"name,noindex" json:"name"`
12         CourseVersion string       `datastore:"course_version,noindex"
           ↪  json:"course_version"`
```

```
13          URL           string       `datastore:"url,noindex" json:"url"`
14          Habilitations []ForeignKey `datastore:"habilitations,noindex"
            ↪ json:"habilitations"`
15  }
```

Arquivo discipline.go

```
1   package models
2
3   const TableDiscipline = "discipline"
4
5   // Workload defines a workload related to a discipline
6   type DisciplineWorkload struct {
7           ID    string  `datastore:"id,noindex" json:"id"`
8           Name  string  `datastore:"name,noindex" json:"name"`
9           Unit  string  `datastore:"unit,noindex" json:"unit"`
10          Value float64 `datastore:"value,noindex" json:"value"`
11  }
12
13  type DisciplineRelated struct {
14          Name string `datastore:"name,noindex" json:"name"`
15          Type string `datastore:"type,noindex" json:"type"`
16  }
17
18  type DisciplineRelatedItem struct {
19          RelatedID   int    `datastore:"related_id,noindex"
            ↪ json:"related_id"`
20          Type        string `datastore:"type,noindex" json:"type"`
21          Value       string `datastore:"value,noindex" json:"value"`
22          Description string `datastore:"description,noindex"
            ↪ json:"description"`
23  }
24
25  type Discipline struct {
26          ID           string                  `datastore:"id,noindex"
            ↪ json:"id"`
27          UniversityID string
            ↪ `datastore:"university_id,noindex" json:"university_id"`
28          Course       ForeignKey              `datastore:"course,noindex"
            ↪ json:"course"`
```

```
29          Habilitation ForeignKey
            ↪ `datastore:"habilitation,noindex" json:"habilitation"`
30          Step         ForeignKey              `datastore:"step,noindex"
            ↪ json:"step"`
31          GenericID    string
            ↪ `datastore:"generic_id,noindex" json:"generic_id"`
32          Version      string                  `datastore:"version,noindex"
            ↪ json:"version"`
33          Code         string                  `datastore:"code,noindex"
            ↪ json:"code"`
34          Name         string                  `datastore:"name,noindex"
            ↪ json:"name"`
35          Description  string
            ↪ `datastore:"description,noindex" json:"description"`
36          Type         string                  `datastore:"type,noindex"
            ↪ json:"type"`
37          Workload     []DisciplineWorkload
            ↪ `datastore:"workload,noindex" json:"workload"`
38          Related      []DisciplineRelated     `datastore:"related,noindex"
            ↪ json:"related"`
39          RelatedItems []DisciplineRelatedItem
            ↪ `datastore:"related_items,noindex" json:"related_items"`
40          Tables       []Table                 `datastore:"tables,noindex"
            ↪ json:"tables"`
41          TableColumns []TableColumn
            ↪ `datastore:"tables_columns,noindex" json:"tables_columns"`
42          TableRows    []TableRow
            ↪ `datastore:"tables_rows,noindex" json:"tables_rows"`
43  }
44
45  // BasicDiscipline represents a Discipline on the search index
46  type BasicDiscipline struct {
47          ID           string                  `datastore:"id,noindex"
            ↪ json:"noindex"`
48          UniversityID string
            ↪ `datastore:"university_id,noindex" json:"university_id"`
49          Course       ForeignKey              `datastore:"course,noindex"
            ↪ json:"course"`
```

```
50          Habilitation ForeignKey
            ↪ `datastore:"habilitation,noindex" json:"habilitation"`
51          Step         ForeignKey              `datastore:"step,noindex"
            ↪ json:"step"`
52          GenericID    string
            ↪ `datastore:"generic_id,noindex" json:"generic_id"`
53          Version      string                  `datastore:"version,noindex"
            ↪ json:"version"`
54          Code         string                  `datastore:"code,noindex"
            ↪ json:"code"`
55          Name         string                  `datastore:"name,noindex"
            ↪ json:"name"`
56          Description  string
            ↪ `datastore:"description,noindex" json:"description"`
57          Type         string                  `datastore:"type,noindex"
            ↪ json:"type"`
58          Related      []DisciplineRelated     `datastore:"related,noindex"
            ↪ json:"related"`
59          RelatedItems []DisciplineRelatedItem
            ↪ `datastore:"related_items,noindex" json:"related_items"`
60  }
```

Arquivo discipline_offer.go

```
1   package models
2
3   // TableDisciplineOffers constains the name of the table for disciplines
    ↪ offers
4   const TableDisciplineOffers = "discipline_offer"
5
6   // DisciplineOfferTeamRoom represents a room partially
7   type DisciplineOfferRoom struct {
8           ID          string `datastore:"id,noindex" json:"id"`
9           GenericID   string `datastore:"generic_id,noindex"
            ↪ json:"generic_id"`
10          Name        string `datastore:"name,noindex" json:"name"`
11          Description string `datastore:"description,noindex"
            ↪ json:"description"`
12          OLCode      string `datastore:"olcode,noindex" json:"olcode"`
13  }
```

```
14
15 // DisciplineOfferTeamScheduleHourMinute defines hour and minute
16 type DisciplineOfferTeamScheduleHourMinute struct {
17         Hour   int8 `datastore:"hour,noindex" json:"hour"`
18         Minute int8 `datastore:"minute,noindex" json:"minute"`
19 }
20
21 // DisciplineOfferchedule represents a schedule on a team with partial
   ↪  data
22 type DisciplineOfferSchedule struct {
23         Start     DisciplineOfferTeamScheduleHourMinute
           ↪  `datastore:"start,noindex" json:"start"`
24         End       DisciplineOfferTeamScheduleHourMinute
           ↪  `datastore:"end,noindex" json:"end"`
25         DayOfWeek int8
           ↪  `datastore:"day_of_week,noindex" json:"day_of_week"`
26 }
27
28 // DisciplineOfferTeacher is a struct that represents a teacher of a
   ↪  discipline
29 type DisciplineOfferTeacher struct {
30         ID        string `datastore:"id,noindex" json:"id"`
31         GenericID string `datastore:"generic_id,noindex"
           ↪  json:"generic_id"`
32         Name      string `datastore:"name,noindex" json:"name"`
33         URL       string `datastore:"url,noindex" json:"url"`
34 }
35
36 // DisciplineOfferTeamTeacher is a struct that represents a teacher of a
   ↪  discipline
37 type DisciplineOfferTeamTeacher struct {
38         TeamID    int `datastore:"team_id,noindex" json:"team_id"`
39         TeacherID int `datastore:"teacher_id,noindex" json:"teacher_id"`
40 }
41
42 // DisciplineOfferTeamSchedule is a struct that represents a schedule of
   ↪  a discipline
43 type DisciplineOfferTeamSchedule struct {
44         TeamID     int `datastore:"team_id,noindex" json:"team_id"`
```

```
45          ScheduleID int `datastore:"schedule_id,noindex"
            ↪ json:"schedule_id"`
46          RoomID     int `datastore:"room_id,noindex" json:"room_id"`
47  }
48
49  // DisciplineOfferTeamTable is a struct that represents a table in a team
50  type DisciplineOfferTeamTable struct {
51          TeamID      int    `datastore:"team_id,noindex" json:"team_id"`
52          ID          string `datastore:"id,noindex" json:"id"`
53          Title       string `datastore:"title,noindex" json:"title"`
54          Description string `datastore:"description,noindex"
            ↪ json:"description"`
55  }
56
57  // DisciplineOfferTeamCourse is a struct that represents a course of a
    ↪ team
58  type DisciplineOfferTeamCourse struct {
59          ID          string `datastore:"id,noindex" json:"id"`
60          Name        string `datastore:"name,noindex" json:"name"`
61          Exclusivity string `datastore:"exclusivity,noindex"
            ↪ json:"exclusivity"`
62  }
63
64  // DisciplineOfferTeamVacancy is a struct that represents the vacancy
    ↪ data of a team
65  type DisciplineOfferTeamVacancy struct {
66          Offered int `datastore:"offered,noindex" json:"offered"`
67          Filled  int `datastore:"filled,noindex" json:"filled"`
68  }
69
70  // DisciplineOfferTeam is a struct that represents a team
71  type DisciplineOfferTeam struct {
72          ID        string                    `datastore:"id,noindex"
            ↪ json:"id"`
73          Code      string                    `datastore:"code,noindex"
            ↪ json:"code"`
74          Vacancies DisciplineOfferTeamVacancy
            ↪ `datastore:"vacancies,noindex" json:"vacancies"`
```

```
75          Course    DisciplineOfferTeamCourse  `datastore:"course,noindex"
            ↪  json:"course"`
76  }
77
78  // DisciplineOffer is a struct that represents a discipline with
79  // teams embedded
80  type DisciplineOffer struct {
81          ID            string
            ↪  `datastore:"id,noindex" json:"id"`
82          UniversityID  string
            ↪  `datastore:"university_id,noindex" json:"university_id"`
83          Period        ForeignKey
            ↪  `datastore:"period,noindex" json:"period"`
84          Campus        ForeignKey
            ↪  `datastore:"campus,noindex" json:"campus"`
85          GenericID     string
            ↪  `datastore:"generic_id,noindex" json:"generic_id"`
86          Version       string
            ↪  `datastore:"version,noindex" json:"version"`
87          Code          string
            ↪  `datastore:"code,noindex" json:"code"`
88          Name          string
            ↪  `datastore:"name,noindex" json:"name"`
89          Teams         []DisciplineOfferTeam
            ↪  `datastore:"teams,noindex" json:"teams"`
90          Schedules     []DisciplineOfferSchedule
            ↪  `datastore:"schedules,noindex" json:"schedules"`
91          Rooms         []DisciplineOfferRoom
            ↪  `datastore:"rooms,noindex" json:"rooms"`
92          Teachers      []DisciplineOfferTeacher
            ↪  `datastore:"teachers,noindex" json:"teachers"`
93          TeamSchedules []DisciplineOfferTeamSchedule
            ↪  `datastore:"team_schedules,noindex" json:"team_schedules"`
94          TeamTeachers  []DisciplineOfferTeamTeacher
            ↪  `datastore:"team_teachers,noindex" json:"team_teachers"`
95          TeamTables    []DisciplineOfferTeamTable
            ↪  `datastore:"team_tables,noindex" json:"team_tables"`
```

```
96          TableColumns  []TableColumn
            ↪  `datastore:"team_tables_columns,noindex"
            ↪  json:"team_tables_columns"`
97          TableRows     []TableRow
            ↪  `datastore:"team_tables_rows,noindex"
            ↪  json:"team_tables_rows"`
98  }
99
100 // BasicDisciplineOffer is a struct that represents a discipline in
    ↪  search index
101 type BasicDisciplineOffer struct {
102         ID           string     `datastore:"id,noindex" json:"id"`
103         UniversityID string     `datastore:"university_id,noindex"
            ↪  json:"university_id"`
104         Period       ForeignKey `datastore:"period,noindex"
            ↪  json:"period"`
105         Campus       ForeignKey `datastore:"campus,noindex"
            ↪  json:"campus"`
106         OfferID      string     `datastore:"offer_id,noindex"
            ↪  json:"offer_id"`
107         Version      string     `datastore:"version,noindex"
            ↪  json:"version"`
108         Code         string     `datastore:"code,noindex" json:"code"`
109         Name         string     `datastore:"name,noindex" json:"name"`
110 }
```

Arquivo foreign.go

```
1  package models
2
3  type ForeignKey struct {
4          ID   string `datastore:"id,noindex" json:"id"`
5          Name string `datastore:"name,noindex" json:"name"`
6  }
```

Arquivo habilitation.go

```
1  package models
2
```

```
3  const TableHabilitations = "habilitation"
4
5  // HabilitationLimit defines the fields neded to represent a limit in a
6  // habilitation
7  type HabilitationLimit struct {
8          ID      string `datastore:"id,noindex" json:"id"`
9          Name    string `datastore:"name,noindex" json:"name"`
10         Context string `datastore:"context,noindex" json:"context"`
11         Unit    string `datastore:"unit,noindex" json:"unit"`
12 }
13
14 // HabilitationLimitValue defines the fields neded to save the values of
   ↪  a limit
15 // in a habilitation
16 type HabilitationLimitValue struct {
17         ID    int     `datastore:"id,noindex" json:"id"`
18         Key   string  `datastore:"key,noindex" json:"key"`
19         Value float64 `datastore:"value,noindex" json:"value"`
20 }
21
22 // HabilitationStep defines the a step of a habilitation
23 type HabilitationStep struct {
24         ID   string `datastore:"id,noindex" json:"id"`
25         Name string `datastore:"name,noindex" json:"name"`
26 }
27
28 // HabilitationStepDiscipline defines a discipline in a step of a
   ↪  habilitation
29 type HabilitationStepDiscipline struct {
30         StepID     int        `datastore:"id,noindex" json:"id"`
31         Discipline ForeignKey `datastore:"discipline,noindex"
           ↪  json:"discipline"`
32 }
33
34 // Habilitation is a habilitation of a course in a university
35 type Habilitation struct {
36         ID              string
           ↪  `datastore:"id,noindex" json:"id"`
```

```
37          UniversityID    string
            ↪ `datastore:"university_id,noindex" json:"university_id"`
38          Course          ForeignKey
            ↪ `datastore:"course,noindex" json:"course"`
39          Version         string
            ↪ `datastore:"version,noindex" json:"version"`
40          Name            string
            ↪ `datastore:"name,noindex" json:"name"`
41          Limits          []HabilitationLimit
            ↪ `datastore:"limits,noindex" json:"limits"`
42          LimitValues     []HabilitationLimitValue
            ↪ `datastore:"limit_values,noindex" json:"limit_values"`
43          Tables          []Table
            ↪ `datastore:"tables,noindex" json:"tables"`
44          TableColumns    []TableColumn
            ↪ `datastore:"teams_tables_columns,noindex"
            ↪ json:"teams_tables_columns"`
45          TableRows       []TableRow
            ↪ `datastore:"teams_tables_rows,noindex"
            ↪ json:"teams_tables_rows"`
46          Steps           []HabilitationStep
            ↪ `datastore:"steps,noindex" json:"steps"`
47          StepDisciplines []HabilitationStepDiscipline
            ↪ `datastore:"step_disciplines,noindex"
            ↪ json:"steps_disciplines"`
48  }
```

Arquivo period.go

```
1   package models
2
3   // TablePeriod contains the name of the table that stores the semesters
4   const TablePeriod = "period"
5
6   // PeriodCampus is a struct that represents a Campus on a Period
7   type PeriodCampus struct {
8           ID     string `datastore:"id,noindex" json:"id"`
9           Name   string `datastore:"name,noindex" json:"name"`
10          OLCode string `datastore:"olcode,noindex" json:"olcode"`
11  }
```

```
12
13 // Period is a struct that represents a period (semester or quarter, for
14 // example) in a University
15 type Period struct {
16         ID           string         `datastore:"id,noindex" json:"id"`
17         UniversityID string         `datastore:"university_id,noindex"
           ↪  json:"university_id"`
18         Version      string         `datastore:"version,noindex"
           ↪  json:"version"`
19         Name         string         `datastore:"name,noindex"
           ↪  json:"name"`
20         Campi        []PeriodCampus `datastore:"campi,noindex"
           ↪  json:"campi"`
21 }
```

Arquivo plan.go

```
1  package models
2
3  import "time"
4
5  const (
6          // TablePlan contains the name of the table that stores the plans
           ↪  of the users
7          TablePlan = "plan"
8          // TablePlanVersion contains the name of the table that stores
           ↪  the versions of the plans of the users
9          TablePlanVersion = "plan_version"
10 )
11
12 type PlanVersionDiscipline struct {
13         ID           string     `datastore:"id,noindex" json:"id"`
14         Course       ForeignKey `datastore:"course,noindex"
           ↪  json:"course"`
15         Habilitation ForeignKey `datastore:"habilitation,noindex"
           ↪  json:"habilitation"`
16         Step         ForeignKey `datastore:"step,noindex" json:"step"`
17         GenericID    string     `datastore:"generic_id,noindex"
           ↪  json:"generic_id"`
```

```
18          Version      string     `datastore:"version,noindex"
            ↪ json:"version"`
19          Code         string     `datastore:"code,noindex" json:"code"`
20          Name         string     `datastore:"name,noindex" json:"name"`
21          Type         string     `datastore:"type,noindex" json:"type"`
22          Description  string     `datastore:"description,noindex"
            ↪ json:"description"`
23  }
24
25  type PlanVersionRelatedDisciplineSummary struct {
26          DisciplineID int        `datastore:"discipline_id,noindex"
            ↪ json:"discipline_id"`
27          ID           string     `datastore:"id,noindex" json:"id"`
28          GenericID    string     `datastore:"generic_id,noindex"
            ↪ json:"generic_id"`
29          Version      string     `datastore:"version,noindex"
            ↪ json:"version"`
30          Course       ForeignKey `datastore:"course,noindex"
            ↪ json:"course"`
31          Habilitation ForeignKey `datastore:"habilitation,noindex"
            ↪ json:"habilitation"`
32          Step         ForeignKey `datastore:"step,noindex" json:"step"`
33          Code         string     `datastore:"code,noindex" json:"code"`
34          Name         string     `datastore:"name,noindex" json:"name"`
35          Type         string     `datastore:"type,noindex" json:"type"`
36          Description  string     `datastore:"description,noindex"
            ↪ json:"description"`
37  }
38
39  type PlanVersionDisciplineRelated struct {
40          DisciplineID int    `datastore:"discipline_id,noindex"
            ↪ json:"discipline_id"`
41          Name         string `datastore:"name,noindex" json:"name"`
42          Type         string `datastore:"type,noindex" json:"type"`
43  }
44
45  type PlanVersionDisciplineOffer struct {
46          ID        string     `datastore:"id,noindex" json:"id"`
```

```
47          GenericID string     `datastore:"generic_id,noindex"
            ↪ json:"generic_id"`
48          Version   string     `datastore:"version,noindex" json:"version"`
49          Campus    ForeignKey `datastore:"campus,noindex" json:"campus"`
50          Code      string     `datastore:"code,noindex" json:"code"`
51          Name      string     `datastore:"name,noindex" json:"name"`
52          Custom    bool       `datastore:"custom,noindex" json:"custom"`
53  }
54
55  type PlanVersionTeam struct {
56          DisciplineOfferID int
            ↪ `datastore:"discipline_offer_id,noindex"
            ↪ json:"discipline_offer_id"`
57          ID                string
            ↪ `datastore:"id,noindex" json:"id"`
58          Version           string
            ↪ `datastore:"version,noindex" json:"version"`
59          Code              string
            ↪ `datastore:"code,noindex" json:"code"`
60          Vacancies         DisciplineOfferTeamVacancy
            ↪ `datastore:"vacancy,noindex" json:"vacancy"`
61          Course            DisciplineOfferTeamCourse
            ↪ `datastore:"course,noindex" json:"course"`
62  }
63
64  type PlanPossibility struct {
65          Name string `datastore:"name,noindex" json:"name"`
66  }
67
68  type PlanPossibilityDiscipline struct {
69          PossibilityID int  `datastore:"id,noindex" json:"id"`
70          DisciplineID  int  `datastore:"discipline_id,noindex"
            ↪ json:"discipline_id"`
71          Enabled       bool `datastore:"enabled,noindex" json:"enabled"`
72  }
73
74  type PlanPossibilityTeam struct {
75          PossibilityID int    `datastore:"id,noindex" json:"id"`
```

```
76          OfferID       int    `datastore:"discipline_offer_id,noindex"
            ↪ json:"discipline_offer_id"`
77          TeamID        int    `datastore:"team_id,noindex" json:"team_id"`
78          Color         string `datastore:"color,noindex" json:"color"`
79          Enabled       bool   `datastore:"enabled,noindex" json:"enabled"`
80          Selected      bool   `datastore:"selected,noindex"
            ↪ json:"selected"`
81  }
82
83  type PlanTableType string
84
85  const (
86          TeamTableType       PlanTableType = "team"
87          DisciplineTableType PlanTableType = "discipline"
88  )
89
90  type PlanTable struct {
91          Type        PlanTableType `datastore:"type,noindex" json:"type"`
92          Index       int           `datastore:"index,noindex"
            ↪ json:"index"`
93          ID          string        `datastore:"id,noindex" json:"id"`
94          Title       string        `datastore:"title,noindex"
            ↪ json:"title"`
95          Description string        `datastore:"description,noindex"
            ↪ json:"description"`
96  }
97
98  // PlanVersion is a struct that represents a version of a Plan, created
    ↪  by a User
99  type PlanVersion struct {
100         ID                          string
            ↪ `datastore:"id,noindex" json:"id"`
101         University                  ForeignKey
            ↪ `datastore:"university,noindex" json:"university"`
102         Period                      ForeignKey
            ↪ `datastore:"period,noindex" json:"period"`
103         Plan                        ForeignKey
            ↪ `datastore:"plan,noindex" json:"plan"`
```

```
104         SavedAt                         time.Time
            ↪ `datastore:"saved_at,noindex" json:"saved_at"`
105         SelectedPossibility             int
            ↪ `datastore:"selected_possibility,noindex"
            ↪ json:"selected_possibility"`
106         TotalPossibilities              int
            ↪ `datastore:"total_possibilities,noindex"
            ↪ json:"total_possibilities"`
107         Possibilities                   []PlanPossibility
            ↪ `datastore:"possibilities,noindex" json:"possibilities"`
108         PossibilitiesTeams              []PlanPossibilityTeam
            ↪ `datastore:"possibilities_teams,noindex"
            ↪ json:"possibilities_teams"`
109         PossibilitiesDisciplines        []PlanPossibilityDiscipline
            ↪ `datastore:"possibilities_disciplines,noindex"
            ↪ json:"possibilities_disciplines"`
110         Disciplines                     []PlanVersionDiscipline
            ↪ `datastore:"disciplines,noindex" json:"disciplines"`
111         DisciplinesRelatedDisciplines
            ↪ []PlanVersionRelatedDisciplineSummary
            ↪ `datastore:"disciplines_related_disciplines,noindex"
            ↪ json:"disciplines_related_disciplines"`
112         DisciplinesRelated              []PlanVersionDisciplineRelated
            ↪ `datastore:"disciplines_related,noindex"
            ↪ json:"disciplines_related"`
113         DisciplinesRelatedItem          []DisciplineRelatedItem
            ↪ `datastore:"disciplines_related_items,noindex"
            ↪ json:"disciplines_related_items"`
114         DisciplinesOffers               []PlanVersionDisciplineOffer
            ↪ `datastore:"discipline_offers,noindex"
            ↪ json:"discipline_offers"`
115         Teams                           []PlanVersionTeam
            ↪ `datastore:"teams,noindex" json:"teams"`
116         Rooms                           []DisciplineOfferRoom
            ↪ `datastore:"rooms,noindex" json:"rooms"`
117         Teachers                        []DisciplineOfferTeacher
            ↪ `datastore:"teachers,noindex" json:"teachers"`
118         Schedules                       []DisciplineOfferSchedule
            ↪ `datastore:"schedules,noindex" json:"schedules"`
```

```
119         TeamTeachers                    []DisciplineOfferTeamTeacher
            ↪ `datastore:"team_teachers,noindex" json:"team_teachers"`
120         TeamSchedules                   []DisciplineOfferTeamSchedule
            ↪ `datastore:"team_schedules,noindex" json:"team_schedules"`
121         Tables                          []PlanTable
            ↪ `datastore:"tables,noindex" json:"tables"`
122         TableColumns                    []TableColumn
            ↪ `datastore:"tables_columns,noindex" json:"tables_columns"`
123         TableRows                       []TableRow
            ↪ `datastore:"tables_rows,noindex" json:"tables_rows"`
124 }
125
126 type PlanForeignVersion struct {
127         ID                string    `datastore:"id,noindex" json:"id"`
128         SavedAt           time.Time `datastore:"saved_at,noindex"
            ↪ json:"saved_at"`
129         TotalPossibilities int
            ↪ `datastore:"total_possibilities,noindex"
            ↪ json:"total_possibilities"`
130 }
131
132 type PlanSchedulePreview struct {
133         ID       string                  `datastore:"id,noindex"
            ↪ json:"id"`
134         Interval DisciplineOfferSchedule `datastore:"interval,noindex"
            ↪ json:"interval"`
135         Title    string                  `datastore:"title,noindex"
            ↪ json:"title"`
136         Color    string                  `datastore:"color,noindex"
            ↪ json:"color"`
137 }
138
139 type PlanVersionSchedule struct {
140         Version  int `datastore:"version,noindex" json:"version"`
141         Schedule int `datastore:"schedule,noindex" json:"schedule"`
142 }
143
144 // Plan is a struct that represents a Plan, created by a User
145 type Plan struct {
```

```
146         ID              string              `datastore:"id,noindex"
            ↪ json:"id"`
147         Name            string              `datastore:"name,noindex"
            ↪ json:"name"`
148         CreatedAt       time.Time
            ↪ `datastore:"created_at,noindex" json:"created_at"`
149         LastModified    time.Time
            ↪ `datastore:"last_modified,noindex" json:"last_modified"`
150         User            ForeignKey          `datastore:"user,noindex"
            ↪ json:"user"`
151         Public          bool
            ↪ `datastore:"public,noindex" json:"public"`
152         University      ForeignKey
            ↪ `datastore:"university,noindex" json:"university"`
153         Period          ForeignKey
            ↪ `datastore:"period,noindex" json:"period"`
154         Versions        []PlanForeignVersion
            ↪ `datastore:"versions,noindex" json:"versions"`
155         VersionSchedules []PlanVersionSchedule
            ↪ `datastore:"versions_schedules,noindex"
            ↪ json:"versions_schedules"`
156         Schedules       []PlanSchedulePreview
            ↪ `datastore:"schedules,noindex" json:"schedules"`
157 }
```

Arquivo room.go

```
1  package models
2
3  // Room represents a room on the search index
4  type Room struct {
5          ID           string     `datastore:"id,noindex" json:"id"`
6          GenericID    string     `datastore:"generic_id,noindex"
           ↪ json:"generic_id"`
7          UniversityID string     `datastore:"university_id,noindex"
           ↪ json:"university_id"`
8          Period       ForeignKey `datastore:"period_id,noindex"
           ↪ json:"period_id"`
9          Campus       ForeignKey `datastore:"campus,noindex"
           ↪ json:"campus"`
```

```
10          Name         string     `datastore:"name,noindex" json:"name"`
11          OLCode       string     `datastore:"olcode,noindex"
            ↪  json:"olcode"`
12          Description  string     `datastore:"description,noindex"
            ↪  json:"description"`
13  }
```

Arquivo table.go

```
1   package models
2
3   // Table is a struct that represents a embedded table
4   type Table struct {
5           ID          string `datastore:"id,noindex" json:"id"`
6           Title       string `datastore:"title,noindex" json:"title"`
7           Description string `datastore:"description,noindex"
            ↪  json:"description"`
8   }
9
10  // TableColumn is a struct that represents a column in a table
11  type TableColumn struct {
12          TableID int    `datastore:"table_id,noindex" json:"table_id"`
13          ID      string `datastore:"id,noindex" json:"id"`
14          Name    string `datastore:"name,noindex" json:"name"`
15  }
16
17  // TableRow represents a row in the table
18  type TableRow struct {
19          Row    int    `datastore:"row,noindex" json:"row"`
20          Column int    `datastore:"column,noindex" json:"column"`
21          Value  string `datastore:"value,noindex" json:"value"`
22  }
```

Arquivo teacher.go

```
1   package models
2
3   // Teacher represents a room on the search index
4   type Teacher struct {
```

```
5           ID           string     `datastore:"id,noindex" json:"id"`
6           GenericID    string     `datastore:"generic_id,noindex"
            ↪ json:"generic_id"`
7           UniversityID string     `datastore:"university_id,noindex"
            ↪ json:"university_id"`
8           Period       ForeignKey `datastore:"period_id,noindex"
            ↪ json:"period_id"`
9           Version      string     `datastore:"version,noindex"
            ↪ json:"version"`
10          Name         string     `datastore:"name,noindex" json:"name"`
11          URL          string     `datastore:"url,noindex" json:"url"`
12  }
```

Arquivo team.go

```
1   package models
2
3   // DisciplineOffer represents a Discipline Offer on the search index
4   type TeamDisciplineOffer struct {
5           ID        string `datastore:"id,noindex" json:"id"`
6           GenericID string `datastore:"generic_id,noindex"
            ↪ json:"generic_id"`
7           Code      string `datastore:"code,noindex" json:"code"`
8           Name      string `datastore:"name,noindex" json:"name"`
9   }
10
11  // CourseOffer represents the course to which the team is focused on
12  // in the search index
13  type TeamCourseOffer struct {
14          ID        string `datastore:"id,noindex" json:"id"`
15          Name      string `datastore:"name,noindex" json:"name"`
16          Exclusive string `datastore:"exclusive,noindex" json:"exclusive"`
17  }
18
19  // Team represents a Team on the search index
20  type Team struct {
21          ID           string              `datastore:"id,noindex"
            ↪ json:"id"`
22          UniversityID string
            ↪ `datastore:"university_id,noindex" json:"university_id"`
```

```
23          PeriodID     string              `datastore:"period_id,noindex"
            ↪ json:"period_id"`
24          Version      string              `datastore:"version,noindex"
            ↪ json:"version"`
25          Discipline   TeamDisciplineOffer `datastore:"discipline,noindex"
            ↪ json:"discipline"`
26          Course       TeamCourseOffer     `datastore:"course,noindex"
            ↪ json:"course"`
27          Code         string              `datastore:"code,noindex"
            ↪ json:"code"`
28  }
```

Arquivo university.go

```
1   package models
2
3   import "time"
4
5   const TableUniversity = "university"
6
7   type UniversityFile struct {
8           ID      string `datastore:"id,noindex" json:"id"`
9           File    string `datastore:"file,noindex" json:"file"`
10          Version int    `datastore:"version,noindex" json:"version"`
11  }
12
13  type UniversityQueueItem struct {
14          ID          string    `datastore:"id,noindex" json:"id"`
15          Filename    string    `datastore:"filename,noindex"
            ↪ json:"filename"`
16          LastUpdated time.Time `datastore:"last_updated,noindex"
            ↪ json:"last_updated"`
17  }
18
19  type UniversityMutex struct {
20          Owner       string    `datastore:"owner,noindex" json:"owner"`
21          LastUpdated time.Time `datastore:"last_updated,noindex"
            ↪ json:"last_updated"`
22  }
23
```

```
24 type UniversityData struct {
25         UserID      string                `datastore:"user_id,noindex"
           ↪ json:"user_id"`
26         BaseURL     string                `datastore:"base_url,noindex"
           ↪ json:"base_url"`
27         DeleteURL   string                `datastore:"delete_url,noindex"
           ↪ json:"delete_url"`
28         AboutURL    string                `datastore:"about_url,noindex"
           ↪ json:"about_url"`
29         Queue       []UniversityQueueItem `datastore:"queue,noindex"
           ↪ json:"queue"`
30         LastUpdated time.Time
           ↪ `datastore:"last_updated,noindex" json:"last_updated"`
31         Mutex       UniversityMutex       `datastore:"mutex,noindex"
           ↪ json:"mutex"`
32         Secret      string                `datastore:"secret,noindex"
           ↪ json:"secret_offers"`
33 }
34
35 // University is a struct that represents a University..
36 type University struct {
37         ID            string           `datastore:"id,noindex" json:"id"`
38         Name          string           `datastore:"name,noindex"
           ↪ json:"name"`
39         Acronym       string           `datastore:"acronym,noindex"
           ↪ json:"acronym"`
40         User          ForeignKey       `datastore:"user,noindex"
           ↪ json:"user"`
41         Public        bool             `datastore:"public,noindex"
           ↪ json:"public"`
42         Offers        UniversityData   `datastore:"offers,noindex"
           ↪ json:"offers"`
43         Habilitations UniversityData   `datastore:"habilitations,noindex"
           ↪ json:"habilitations"`
44         Courses       []ForeignKey     `datastore:"courses,noindex"
           ↪ json:"courses"`
45         Periods       []ForeignKey     `datastore:"periods,noindex"
           ↪ json:"periods"`
```

```
46          Files          []UniversityFile `datastore:"files,noindex"
            ↪  json:"files"`
47  }
48
49  func (u *University) GetLatestFile(id string) UniversityFile {
50          var result UniversityFile
51          for _, file := range u.Files {
52                  if file.ID == id {
53                          result = file
54                  }
55          }
56          return result
57  }
58
59  func (u *University) AddFile(id, file string) {
60          last := u.GetLatestFile(id)
61          nextVersion := 0
62          if last.ID == id {
63                  nextVersion = last.Version + 1
64          }
65          toAdd := UniversityFile{
66                  ID:      id,
67                  Version: nextVersion,
68                  File:    file,
69          }
70          u.Files = append([]UniversityFile{toAdd}, u.Files...)
71  }
```

Arquivo user.go

```
1   package models
2
3   import (
4           "time"
5
6           "github.com/volatiletech/authboss"
7           "github.com/volatiletech/authboss/otp/twofactor/totp2fa"
8   )
9
10  const (
```

```
11          TableUser                = "user"
12          TableUserRecoverSelector = "user_recover_selector"
13          TableUserConfirmSelector = "user_confirm_selector"
14  )
15
16  type UserSelector struct {
17          UserID       string    `datastore:"user_id,noindex"
            ↪  json:"user_id"`
18          Selector     string    `datastore:"selector,noindex"
            ↪  json:"selector"`
19          RegisteredAt time.Time `datastore:"registered_at,noindex"
            ↪  json:"registered_at"`
20  }
21
22  type UserPlan struct {
23          ID           string     `datastore:"id,noindex" json:"id"`
24          Name         string     `datastore:"name,noindex" json:"name"`
25          University   ForeignKey `datastore:"university,noindex"
            ↪  json:"university"`
26          Period       ForeignKey `datastore:"period,noindex"
            ↪  json:"period"`
27          CreatedAt    time.Time  `datastore:"created_at,noindex"
            ↪  json:"created_at"`
28          LastModified time.Time  `datastore:"last_modified,noindex"
            ↪  json:"last_modified"`
29  }
30
31  type UserDiscipline struct {
32          ID        string `datastore:"id,noindex" json:"id"`
33          GenericID string `datastore:"generic_id,noindex"
            ↪  json:"generic_id"`
34          Code      string `datastore:"code,noindex" json:"code"`
35          Name      string `datastore:"name,noindex" json:"name"`
36  }
37
38  type UserUniversity struct {
39          ID      string `datastore:"id,noindex" json:"id"`
40          Name    string `datastore:"name,noindex" json:"name"`
41          Acronym string `datastore:"acronym,noindex" json:"acronym"`
```

```
42          Public  bool   `datastore:"public,noindex" json:"public"`
43  }
44
45  type User struct {
46          // Auth
47          ID       string `datastore:"id,noindex" json:"id"` // E-mail or
            ↪  ID from OAuth 2 provider
48          Password string `datastore:"password,noindex" json:"password"`
49          Email    string `datastore:"email,noindex" json:"email"`
50
51          // OAuth2 Auth
52          OAuth2UID          string    `datastore:"oauth2_uid,noindex"
            ↪  json:"oauth2_uid"`
53          OAuth2Provider     string    `datastore:"oauth2_provider,noindex"
            ↪  json:"oauth2_provider"`
54          OAuth2AccessToken  string
            ↪  `datastore:"oauth2_access_token,noindex"
            ↪  json:"oauth2_access_token"`
55          OAuth2RefreshToken string
            ↪  `datastore:"oauth2_refresh_token,noindex"
            ↪  json:"oauth2_refresh_token"`
56          OAuth2Expiry       time.Time `datastore:"oauth2_expiry,noindex"
            ↪  json:"oauth2_expiry"`
57
58          // Confirm data
59          Confirmed       bool   `datastore:"confirmed,noindex"
            ↪  json:"confirmed"`
60          ConfirmSelector string `datastore:"confirm_selector,noindex"
            ↪  json:"confirm_selector"`
61          ConfirmVerifier string `datastore:"confirm_verifier,noindex"
            ↪  json:"confirm_verifier"`
62
63          // Lock
64          AttemptCount int       `datastore:"attempt_count,noindex"
            ↪  json:"attempt_count"`
65          LastAttempt  time.Time `datastore:"last_attempt,noindex"
            ↪  json:"last_attempt"`
66          Locked       time.Time `datastore:"locked,noindex" json:"locked"`
67
```

```
68          // 2fa data
69          TOTPSecretKey string `datastore:"totp_secret_key"
            ↪  json:"totp_secret_key"`
70
71          // Recovery  data
72          RecoveryCodes   string    `datastore:"recovery_codes,noindex"
            ↪  json:"recovery_codes"`
73          RecoverSelector string    `datastore:"recover_selector,noindex"
            ↪  json:"recover_selector"`
74          RecoverVerifier string    `datastore:"recover_verifier,noindex"
            ↪  json:"recover_verifier"`
75          RecoverExpiry   time.Time `datastore:"recover_expiry,noindex"
            ↪  json:"recover_expiry"`
76
77          // App specific data
78          Name                  string           `datastore:"name,noindex"
            ↪  json:"name"`
79          CompletedDisciplines []UserDiscipline
            ↪  `datastore:"completed_disciplines,noindex"
            ↪  json:"completed_disciplines"`
80          University           ForeignKey
            ↪  `datastore:"university,noindex" json:"university"`
81          Courses              []ForeignKey
            ↪  `datastore:"courses,noindex" json:"courses"`
82          Habilitations        []ForeignKey
            ↪  `datastore:"habilitations,noindex" json:"habilitations"`
83          Plans                []UserPlan       `datastore:"plans,noindex"
            ↪  json:"plans"`
84          Universities         []UserUniversity
            ↪  `datastore:"universities,noindex" json:"universities"`
85          RememberTokens       []string
            ↪  `datastore:"remember_tokens,noindex" json:"remember_tokens"`
86  }
87
88  func (u *User) GetEmail() string {
89          if u.Email == "" {
90                  return u.ID
91          }
92          return u.Email
```

```
93  }
94
95  func (u *User) PutEmail(email string) {
96          u.Email = email
97  }
98
99  func (u *User) GetConfirmed() bool {
100         return u.Confirmed
101 }
102
103 func (u *User) PutConfirmed(confirmed bool) {
104         u.Confirmed = confirmed
105 }
106
107 func (u *User) GetConfirmSelector() string {
108         return u.ConfirmSelector
109 }
110
111 func (u *User) PutConfirmSelector(confirmSelector string) {
112         u.ConfirmSelector = confirmSelector
113 }
114
115 func (u *User) GetConfirmVerifier() string {
116         return u.ConfirmVerifier
117 }
118
119 func (u *User) PutConfirmVerifier(confirmVerifier string) {
120         u.ConfirmVerifier = confirmVerifier
121 }
122
123 func (u *User) GetAttemptCount() int {
124         return u.AttemptCount
125 }
126
127 func (u *User) PutAttemptCount(attemptCount int) {
128         u.AttemptCount = attemptCount
129 }
130
131 func (u *User) GetLastAttempt() time.Time {
```

```
132         return u.LastAttempt
133 }
134
135 func (u *User) PutLastAttempt(lastAttempt time.Time) {
136         u.LastAttempt = lastAttempt
137 }
138
139 func (u *User) GetLocked() time.Time {
140         return u.Locked
141 }
142
143 func (u *User) PutLocked(locked time.Time) {
144         u.Locked = locked
145 }
146
147 func (u *User) GetRecoverSelector() string {
148         return u.RecoverSelector
149 }
150
151 func (u *User) PutRecoverSelector(selector string) {
152         u.RecoverSelector = selector
153 }
154
155 func (u *User) GetRecoverVerifier() string {
156         return u.RecoverVerifier
157 }
158
159 func (u *User) PutRecoverVerifier(verifier string) {
160         u.RecoverVerifier = verifier
161 }
162
163 func (u *User) GetRecoverExpiry() time.Time {
164         return u.RecoverExpiry
165 }
166
167 func (u *User) PutRecoverExpiry(expiry time.Time) {
168         u.RecoverExpiry = expiry
169 }
170
```

```
171 func (u *User) GetPassword() string {
172         return u.Password
173 }
174
175 func (u *User) PutPassword(password string) {
176         u.Password = password
177 }
178
179 func (u *User) GetPID() string {
180         return u.ID
181 }
182
183 func (u *User) PutPID(pid string) {
184         u.ID = pid
185 }
186
187 func (u *User) PutOAuth2UID(uid string) {
188         u.OAuth2UID = uid
189 }
190
191 func (u *User) GetOAuth2UID() string {
192         return u.OAuth2UID
193 }
194
195 func (u *User) IsOAuth2User() bool {
196         return len(u.OAuth2UID) != 0
197 }
198
199 func (u *User) PutOAuth2Provider(provider string) {
200         u.OAuth2Provider = provider
201 }
202
203 func (u *User) GetOAuth2Provider() string {
204         return u.OAuth2Provider
205 }
206
207 func (u *User) PutOAuth2AccessToken(token string) {
208         u.OAuth2AccessToken = token
209 }
```

```
210
211 func (u *User) GetOAuth2AccessToken() string {
212         return u.OAuth2AccessToken
213 }
214
215 func (u *User) PutOAuth2RefreshToken(refreshToken string) {
216         u.OAuth2RefreshToken = refreshToken
217 }
218
219 func (u *User) GetOAuth2RefreshToken() string {
220         return u.OAuth2RefreshToken
221 }
222
223 func (u *User) PutOAuth2Expiry(expiry time.Time) {
224         u.OAuth2Expiry = expiry
225 }
226
227 func (u *User) GetOAuth2Expiry() time.Time {
228         return u.OAuth2Expiry
229 }
230
231 func (u *User) GetArbitrary() map[string]string {
232         return map[string]string{
233                 "name": u.Name,
234         }
235 }
236
237 func (u *User) PutArbitrary(arbitrary map[string]string) {
238         if val, ok := arbitrary["email"]; ok {
239                 u.Email = val
240         }
241         if val, ok := arbitrary["name"]; ok {
242                 u.Name = val
243         }
244 }
245
246 func (u *User) GetRecoveryCodes() string {
247         return u.RecoveryCodes
248 }
```

```
249
250 func (u *User) PutRecoveryCodes(codes string) {
251         u.RecoveryCodes = codes
252 }
253
254 func (u *User) GetTOTPSecretKey() string {
255         return u.TOTPSecretKey
256 }
257
258 func (u *User) PutTOTPSecretKey(secretKey string) {
259         u.TOTPSecretKey = secretKey
260 }
261
262 var (
263         assertUser                          = &User{}
264         _           totp2fa.User            = assertUser
265         _           authboss.User           = assertUser
266         _           authboss.AuthableUser   = assertUser
267         _           authboss.ConfirmableUser = assertUser
268         _           authboss.LockableUser   = assertUser
269         _           authboss.RecoverableUser = assertUser
270         _           authboss.ArbitraryUser  = assertUser
271         _           authboss.OAuth2User     = assertUser
272 )
```

B.1.6 Pasta search

Arquivo exact.go

```
1 package search
2
3 import "strings"
4
5 func IsExact(field string) bool {
```

```
6         return field != "default" && (strings.HasSuffix(field, "id") ||
          ↪  strings.HasPrefix(field, "schedule_") ||
          ↪  strings.HasPrefix(field, "related_") ||
          ↪  strings.HasPrefix(field, "has_") || strings.HasPrefix(field,
          ↪  "depends_on") || strings.HasPrefix(field, "equivalent_to") ||
          ↪  strings.HasPrefix(field, "set_with") || field == "type" ||
          ↪  field == "full")
7 }
```

Arquivo related.go

```
1  package search
2
3  import (
4         "github.com/fjorgemota/gurudamatricula/graphql"
5
6         "github.com/fjorgemota/gurudamatricula/models"
7         "github.com/fjorgemota/gurudamatricula/util/indexer"
8  )
9
10 type Options struct {
11        Filter               graphql.SearchFilter
12        Field                string
13        Courses              []models.ForeignKey
14        CompletedDisciplines []models.UserDiscipline
15        Version              string
16 }
17
18 var zero struct{}
19
20 func findDone(index indexer.Index, mapping map[string]struct{}, options
   ↪  Options) (map[string]struct{}, string, error) {
21        expression := indexer.Or{}
22        for _, discipline := range options.CompletedDisciplines {
23                expression.Operations = append(expression.Operations,
                  ↪  indexer.FieldTerminal{
24                        Attribute: "related_equivalent_to",
25                        Value:     discipline.ID,
26                }, indexer.FieldTerminal{
27                        Attribute: "related_equivalent_to",
```

```
28                      Value:     discipline.GenericID,
29              })
30      }
31      it := index.SearchWithOptions(expression, indexer.SearchOptions{
32              Version: options.Version,
33      })
34      var err error
35      for err == nil {
36              var discipline models.BasicDiscipline
37              err = it.Next(&discipline)
38              if err == nil {
39                      for i, related := range discipline.Related {
40                              if related.Type == "equivalent" {
41                                      match := true
42                                      for _, item := range
                                        ↪  discipline.RelatedItems {
43                                              if item.RelatedID == i {
44                                                      if _, ok :=
                                                        ↪  mapping[item.Value];
                                                        ↪  !ok {
45                                                              match =
                                                                ↪  false
46                                                      }
47                                              }
48                                      }
49                                      if match {
50                                              mapping[discipline.ID] =
                                                ↪  zero
51                                              mapping[discipline.GenericID]
                                                ↪  = zero
52                                              break
53                                      }
54                              }
55                      }
56              }
57      }
58      if indexer.IsDoneError(err) {
59              err = nil
60      }
```

```
61          return mapping, it.Info().Query.Version, err
62  }
63
64  func getFullfilledDisciplines(index indexer.Index, mapping
    ↪  map[string]struct{}, options Options) (indexer.Expression, string,
    ↪  error) {
65          var err error
66          var version string
67          mapping, version, err = findDone(index, mapping, options)
68          exp := indexer.Or{}
69          exp.Operations = append(exp.Operations, indexer.FieldTerminal{
70                  Attribute: "has_prerequisite",
71                  Value:     "no",
72          })
73          for key := range mapping {
74                  exp.Operations = append(exp.Operations,
                    ↪  indexer.FieldTerminal{
75                          Attribute: "related_depends_on",
76                          Value:     key,
77                  })
78          }
79          mappingResult := make(map[string]struct{})
80          if err == nil {
81                  it := index.SearchWithOptions(exp, indexer.SearchOptions{
82                          Version: version,
83                  })
84                  for err == nil {
85                          var discipline models.BasicDiscipline
86                          err = it.Next(&discipline)
87                          if err == nil {
88                                  noPrerequisite := true
89                                  for i, related := range
                                    ↪  discipline.Related {
90                                          if related.Type == "prerequisite"
                                            ↪  {
91                                                  noPrerequisite = false
92                                                  match := true
```

```
93                                                  for _, item := range
                                                    ↪  discipline.RelatedItems
                                                    ↪  {
94                                                          if item.RelatedID
                                                            ↪  == i {
95                                                                  if _, ok
                                                                    ↪  :=
                                                                    ↪  mapping[item.Va
                                                                    ↪  !ok {
96                                                                      match
                                                                        ↪  =
                                                                        ↪  false
97                                                                  }
98                                                          }
99                                                  }
100                                                 if _, ok :=
                                                    ↪  mapping[discipline.GenericID];
                                                    ↪  !ok && match {
101                                                         mappingResult[discipline.Ge
                                                            ↪  = zero
102                                                         break
103                                                 }
104                                         }
105                                 }
106                                 if _, ok := mapping[discipline.GenericID];
                                    ↪  !ok && noPrerequisite {
107                                         mappingResult[discipline.GenericID]
                                            ↪  = zero
108                                 }
109                         }
110                 }
111                 if indexer.IsDoneError(err) {
112                         err = nil
113                 }
114         }
115         result := indexer.Or{}
116         for id := range mappingResult {
117                 result.Operations = append(result.Operations,
                    ↪  indexer.FieldTerminal{
```

```
118                         Attribute: options.Field,
119                         Value:     id,
120                 })
121         }
122         return result, version, err
123 }
124
125 func getDisciplinesByCourse(index indexer.Index, mapping
    ↪  map[string]struct{}, options Options) (indexer.Expression, string,
    ↪  error) {
126         var err error
127         expression := indexer.Or{}
128         for _, course := range options.Courses {
129                 expression.Operations = append(expression.Operations,
                    ↪  indexer.FieldTerminal{
130                         Attribute: "course_id",
131                         Value:     course.ID,
132                 })
133         }
134         mappingResult := make(map[string]struct{})
135         it := index.SearchWithOptions(expression, indexer.SearchOptions{
136                 Version: options.Version,
137         })
138         for err == nil {
139                 var discipline models.BasicDiscipline
140                 err = it.Next(&discipline)
141                 if _, ok := mapping[discipline.GenericID]; !ok && err ==
                    ↪  nil {
142                         mappingResult[discipline.GenericID] = zero
143                 }
144         }
145         if indexer.IsDoneError(err) {
146                 err = nil
147         }
148         result := indexer.Or{}
149         for id := range mappingResult {
150                 result.Operations = append(result.Operations,
                    ↪  indexer.FieldTerminal{
151                         Attribute: options.Field,
```

```
152                         Value:     id,
153                 })
154         }
155         return result, it.Info().Query.Version, err
156 }
157
158 func GetAllowedDisciplines(index indexer.Index, options Options)
    ↪  (indexer.Expression, string, error) {
159         mapping := make(map[string]struct{},
            ↪  len(options.CompletedDisciplines)*2)
160         for _, discipline := range options.CompletedDisciplines {
161                 mapping[discipline.ID] = zero
162                 mapping[discipline.GenericID] = zero
163         }
164         expression := indexer.And{}
165         var err error
166         version := options.Version
167         if options.Filter == graphql.SearchFilterPrerequisites ||
            ↪  options.Filter == graphql.SearchFilterAll {
168                 var result indexer.Expression
169                 result, version, err = getFullfilledDisciplines(index,
                    ↪  mapping, options)
170                 if err == nil {
171                         expression.Operations =
                            ↪  append(expression.Operations, result)
172                 }
173         }
174
175         if options.Filter == graphql.SearchFilterCourses ||
            ↪  options.Filter == graphql.SearchFilterAll {
176                 var result indexer.Expression
177                 result, version, err = getDisciplinesByCourse(index,
                    ↪  mapping, options)
178                 if err == nil {
179                         expression.Operations =
                            ↪  append(expression.Operations, result)
180                 }
181         }
182         var resultExpression indexer.Expression = indexer.All{}
```

```
183         if len(expression.Operations) > 0 {
184                 resultExpression = expression
185         }
186         return resultExpression, version, err
187 }
```

B.1.7 Pasta graphql/dataloaders

Arquivo base.go

```
1  package dataloaders
2
3  import (
4          "context"
5          "net/http"
6          "time"
7
8          "github.com/fjorgemota/gurudamatricula/util/kv"
9  )
10
11 type Handler interface {
12         GetStore(ctx context.Context) (kv.Store, error)
13         GetContext(req *http.Request) context.Context
14 }
15
16 type Options struct {
17         // Wait is how long wait before sending a batch
18         Wait time.Duration
19
20         // MaxBatch will limit the maximum number of keys to send in one
           ↪  batch, 0 = not limit
21         MaxBatch int
22 }
23
24 func getDefaultOptions(options Options) Options {
25         if options.Wait == 0 {
26                 options.Wait = 1 * time.Millisecond
27         }
28         if options.MaxBatch == 0 {
29                 options.MaxBatch = 100
```

```
30          }
31          return options
32  }
33
34  //go:generate dataloaden PlanLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Plan
35  //go:generate dataloaden PlanVersionLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.PlanVersion
36  //go:generate dataloaden UserLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.User
37  //go:generate dataloaden DisciplineOfferLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.DisciplineOffer
38  //go:generate dataloaden TeamLoader
    ↪  github.com/fjorgemota/gurudamatricula/models.Team
    ↪  *github.com/fjorgemota/gurudamatricula/graphql.Team
39  //go:generate dataloaden CampusLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Campus
40  //go:generate dataloaden PeriodLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Period
41  //go:generate dataloaden DisciplineLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Discipline
42  //go:generate dataloaden HabilitationLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Habilitation
43  //go:generate dataloaden CourseLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.Course
44  //go:generate dataloaden UniversityLoader string
    ↪  *github.com/fjorgemota/gurudamatricula/models.University
```

Arquivo campus_loader.go

```
1   package dataloaders
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/util/kv"
6   )
7
8   func getCampusLoaderFromStore(store kv.Store, err error, options Options)
    ↪  *CampusLoader {
9           return NewCampusLoader(CampusLoaderConfig{
```

```
10                MaxBatch: options.MaxBatch,
11                Wait:     options.Wait,
12                Fetch: func(keys []string) ([]*models.Campus, []error) {
13                        data := make(map[string]*models.Campus,
                          ↪  len(keys))
14                        result := make([]*models.Campus, len(keys))
15                        errors := make([]error, len(keys))
16                        var queryErrors map[string]error
17                        if err == nil {
18                                queryErrors =
                                  ↪  store.GetMulti(models.TableCampus,
                                  ↪  keys, data)
19                        }
20                        for i, key := range keys {
21                                errors[i] = err
22                                if errors[i] == nil {
23                                        errors[i] = queryErrors[key]
24                                }
25                                if errors[i] == nil {
26                                        result[i] = data[key]
27                                }
28                        }
29                        return result, errors
30                },
31        })
32 }
```

Arquivo course_loader.go

```
1  package dataloaders
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/util/kv"
6  )
7
8  func getCourseLoaderFromStore(store kv.Store, err error, options Options)
   ↪  *CourseLoader {
9          return NewCourseLoader(CourseLoaderConfig{
10                MaxBatch: options.MaxBatch,
```

```
11                  Wait:     options.Wait,
12                  Fetch: func(keys []string) ([]*models.Course, []error) {
13                          data := make(map[string]*models.Course,
                            ↪  len(keys))
14                          result := make([]*models.Course, len(keys))
15                          errors := make([]error, len(keys))
16                          var queryErrors map[string]error
17                          if err == nil {
18                                  queryErrors =
                                    ↪  store.GetMulti(models.TableCourse,
                                    ↪  keys, data)
19                          }
20                          for i, key := range keys {
21                                  errors[i] = err
22                                  if errors[i] == nil {
23                                          errors[i] = queryErrors[key]
24                                  }
25                                  if errors[i] == nil {
26                                          result[i] = data[key]
27                                  }
28                          }
29                          return result, errors
30                  },
31          })
32  }
```

Arquivo discipline_loader.go

```
1   package dataloaders
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/util/kv"
6   )
7
8   func getDisciplineLoaderFromStore(store kv.Store, err error, options
    ↪  Options) *DisciplineLoader {
9           return NewDisciplineLoader(DisciplineLoaderConfig{
10                  MaxBatch: options.MaxBatch,
11                  Wait:     options.Wait,
```

```
12                  Fetch: func(keys []string) ([]*models.Discipline,
                    ↪  []error) {
13                          data := make(map[string]*models.Discipline,
                            ↪  len(keys))
14                          result := make([]*models.Discipline, len(keys))
15                          errors := make([]error, len(keys))
16                          var queryErrors map[string]error
17                          if err == nil {
18                                  queryErrors =
                                    ↪  store.GetMulti(models.TableDiscipline,
                                    ↪  keys, data)
19                          }
20                          for i, key := range keys {
21                                  errors[i] = err
22                                  if errors[i] == nil {
23                                          errors[i] = queryErrors[key]
24                                  }
25                                  if errors[i] == nil {
26                                          result[i] = data[key]
27                                  }
28                          }
29                          return result, errors
30                  },
31          })
32  }
```

Arquivo discipline_offer_loader.go

```
1   package dataloaders
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/util/kv"
6   )
7
8   func getDisciplineOfferLoaderFromStore(store kv.Store, err error, options
    ↪  Options) *DisciplineOfferLoader {
9           return NewDisciplineOfferLoader(DisciplineOfferLoaderConfig{
10                  MaxBatch: options.MaxBatch,
11                  Wait:     options.Wait,
```

```
12                  Fetch: func(keys []string) ([]*models.DisciplineOffer,
                    ↪  []error) {
13                          data := make(map[string]*models.DisciplineOffer,
                            ↪  len(keys))
14                          result := make([]*models.DisciplineOffer,
                            ↪  len(keys))
15                          errors := make([]error, len(keys))
16                          var queryErrors map[string]error
17                          if err == nil {
18                                  queryErrors =
                                    ↪  store.GetMulti(models.TableDisciplineOffers,
                                    ↪  keys, data)
19                          }
20                          for i, key := range keys {
21                                  errors[i] = err
22                                  if err == nil {
23                                          errors[i] = queryErrors[key]
24                                  }
25                                  if errors[i] == nil {
26                                          result[i] = data[key]
27                                  }
28                          }
29                          return result, errors
30                  },
31          })
32  }
```

Arquivo habilitation_loader.go

```
1   package dataloaders
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/util/kv"
6   )
7
8   func getHabilitationLoaderFromStore(store kv.Store, err error, options
    ↪  Options) *HabilitationLoader {
9           return NewHabilitationLoader(HabilitationLoaderConfig{
10                  MaxBatch: options.MaxBatch,
```

```
                Wait:     options.Wait,
                Fetch: func(keys []string) ([]*models.Habilitation,
                ↪  []error) {
                        data := make(map[string]*models.Habilitation,
                        ↪  len(keys))
                        result := make([]*models.Habilitation, len(keys))
                        errors := make([]error, len(keys))
                        var queryErrors map[string]error
                        if err == nil {
                                queryErrors =
                                ↪  store.GetMulti(models.TableHabilitations,
                                ↪  keys, data)
                        }
                        for i, key := range keys {
                                errors[i] = err
                                if errors[i] == nil {
                                        errors[i] = queryErrors[key]
                                }
                                if errors[i] == nil {
                                        result[i] = data[key]
                                }
                        }
                        return result, errors
                },
        })
}
```

Arquivo middleware.go

```
package dataloaders

import (
        "context"
        "net/http"
)

type ctxType string

var loadersType ctxType = "loaders-type"

```

```
12 type contextLoaders struct {
13         universityLoader      *UniversityLoader
14         courseLoader          *CourseLoader
15         planLoader            *PlanLoader
16         planVersionLoader     *PlanVersionLoader
17         disciplineLoader      *DisciplineLoader
18         periodLoader          *PeriodLoader
19         habilitationLoader    *HabilitationLoader
20         disciplineOfferLoader *DisciplineOfferLoader
21         campusLoader          *CampusLoader
22         teamLoader            *TeamLoader
23         userLoader            *UserLoader
24 }
25
26 type middleware struct {
27         handler Handler
28         options Options
29         next    http.Handler
30 }
31
32 func (m middleware) ServeHTTP(resp http.ResponseWriter, req
   ↪  *http.Request) {
33         ctx := m.handler.GetContext(req)
34         store, err := m.handler.GetStore(ctx)
35         ctx = context.WithValue(ctx, loadersType, &contextLoaders{
36                 universityLoader:      getUniversityLoaderFromStore(store,
                  ↪  err, m.options),
37                 courseLoader:          getCourseLoaderFromStore(store,
                  ↪  err, m.options),
38                 planLoader:            getPlanLoaderFromStore(store, err,
                  ↪  m.options),
39                 planVersionLoader:
                  ↪  getPlanVersionLoaderFromStore(store, err, m.options),
40                 disciplineLoader:      getDisciplineLoaderFromStore(store,
                  ↪  err, m.options),
41                 periodLoader:          getPeriodLoaderFromStore(store,
                  ↪  err, m.options),
```

```
42                  habilitationLoader:
                    ↪ getHabilitationLoaderFromStore(store, err,
                    ↪ m.options),
43                  disciplineOfferLoader:
                    ↪ getDisciplineOfferLoaderFromStore(store, err,
                    ↪ m.options),
44                  campusLoader:          getCampusLoaderFromStore(store,
                    ↪ err, m.options),
45                  teamLoader:            getTeamLoaderFromStore(store, err,
                    ↪ m.options),
46                  userLoader:            getUserLoaderFromStore(store, err,
                    ↪ m.options),
47          })
48          req = req.WithContext(ctx)
49          m.next.ServeHTTP(resp, req)
50  }
51
52  func Middleware(next http.Handler, handler Handler, options Options)
    ↪ http.Handler {
53          return middleware{
54                  next:    next,
55                  handler: handler,
56                  options: getDefaultOptions(options),
57          }
58  }
59
60  func getLoaders(ctx context.Context) *contextLoaders {
61          return ctx.Value(loadersType).(*contextLoaders)
62  }
63
64  func GetCampusLoader(ctx context.Context) *CampusLoader {
65          return getLoaders(ctx).campusLoader
66  }
67
68  func GetCourseLoader(ctx context.Context) *CourseLoader {
69          return getLoaders(ctx).courseLoader
70  }
71
72  func GetDisciplineLoader(ctx context.Context) *DisciplineLoader {
```

```
73          return getLoaders(ctx).disciplineLoader
74  }
75
76  func GetDisciplineOfferLoader(ctx context.Context) *DisciplineOfferLoader
    ↪ {
77          return getLoaders(ctx).disciplineOfferLoader
78  }
79
80  func GetHabilitationLoader(ctx context.Context) *HabilitationLoader {
81          return getLoaders(ctx).habilitationLoader
82  }
83
84  func GetPeriodLoader(ctx context.Context) *PeriodLoader {
85          return getLoaders(ctx).periodLoader
86  }
87
88  func GetPlanLoader(ctx context.Context) *PlanLoader {
89          return getLoaders(ctx).planLoader
90  }
91
92  func GetPlanVersionLoader(ctx context.Context) *PlanVersionLoader {
93          return getLoaders(ctx).planVersionLoader
94  }
95
96  func GetTeamLoader(ctx context.Context) *TeamLoader {
97          return getLoaders(ctx).teamLoader
98  }
99
100 func GetUniversityLoader(ctx context.Context) *UniversityLoader {
101         return getLoaders(ctx).universityLoader
102 }
103
104 func GetUserLoader(ctx context.Context) *UserLoader {
105         return getLoaders(ctx).userLoader
106 }
```

Arquivo period_loader.go

```
1   package dataloaders
2
```

```
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/util/kv"
6  )
7
8  func getPeriodLoaderFromStore(store kv.Store, err error, options Options)
   ↪  *PeriodLoader {
9          return NewPeriodLoader(PeriodLoaderConfig{
10                 MaxBatch: options.MaxBatch,
11                 Wait:     options.Wait,
12                 Fetch: func(keys []string) ([]*models.Period, []error) {
13                         data := make(map[string]*models.Period,
                           ↪  len(keys))
14                         result := make([]*models.Period, len(keys))
15                         errors := make([]error, len(keys))
16                         var queryErrors map[string]error
17                         if err == nil {
18                                 queryErrors =
                                   ↪  store.GetMulti(models.TablePeriod,
                                   ↪  keys, data)
19                         }
20                         for i, key := range keys {
21                                 errors[i] = err
22                                 if errors[i] == nil {
23                                         errors[i] = queryErrors[key]
24                                 }
25                                 if errors[i] == nil {
26                                         result[i] = data[key]
27                                 }
28                         }
29                         return result, errors
30                 },
31         })
32 }
```

Arquivo plan_loader.go

```
1  package dataloaders
2
3  import (
```

```
4         "github.com/fjorgemota/gurudamatricula/models"
5         "github.com/fjorgemota/gurudamatricula/util/kv"
6 )
7
8 func getPlanLoaderFromStore(store kv.Store, err error, options Options)
  ↪  *PlanLoader {
9         return NewPlanLoader(PlanLoaderConfig{
10                MaxBatch: options.MaxBatch,
11                Wait:     options.Wait,
12                Fetch: func(keys []string) ([]*models.Plan, []error) {
13                        data := make(map[string]*models.Plan, len(keys))
14                        result := make([]*models.Plan, len(keys))
15                        errors := make([]error, len(keys))
16                        var queryErrors map[string]error
17                        if err == nil {
18                                queryErrors =
                                  ↪  store.GetMulti(models.TablePlan, keys,
                                  ↪  data)
19                        }
20                        for i, key := range keys {
21                                errors[i] = err
22                                if errors[i] == nil {
23                                        errors[i] = queryErrors[key]
24                                }
25                                if errors[i] == nil {
26                                        result[i] = data[key]
27                                }
28                        }
29                        return result, errors
30                },
31        })
32 }
```

Arquivo plan_version_loader.go

```
1 package dataloaders
2
3 import (
4         "github.com/fjorgemota/gurudamatricula/models"
5         "github.com/fjorgemota/gurudamatricula/util/kv"
```

```
)

func getPlanVersionLoaderFromStore(store kv.Store, err error, options
↪  Options) *PlanVersionLoader {
        return NewPlanVersionLoader(PlanVersionLoaderConfig{
                MaxBatch: options.MaxBatch,
                Wait:     options.Wait,
                Fetch: func(keys []string) ([]*models.PlanVersion,
                ↪  []error) {
                        data := make(map[string]*models.PlanVersion,
                        ↪  len(keys))
                        result := make([]*models.PlanVersion, len(keys))
                        errors := make([]error, len(keys))
                        var queryErrors map[string]error
                        if err == nil {
                                queryErrors =
                                ↪  store.GetMulti(models.TablePlanVersion,
                                ↪  keys, data)
                        }
                        for i, key := range keys {
                                errors[i] = err
                                if errors[i] == nil {
                                        errors[i] = queryErrors[key]
                                }
                                if errors[i] == nil {
                                        result[i] = data[key]
                                }
                        }
                        return result, errors
                },
        })
}
```

Arquivo team_loader.go

```
package dataloaders

import (
        "github.com/fjorgemota/gurudamatricula/graphql"
        "github.com/fjorgemota/gurudamatricula/models"
```

```
6         "github.com/fjorgemota/gurudamatricula/util/kv"
7 )
8
9 func getTeamLoaderFromStore(store kv.Store, err error, options Options)
  ↪  *TeamLoader {
10        return NewTeamLoader(TeamLoaderConfig{
11                MaxBatch: options.MaxBatch,
12                Wait:     options.Wait,
13                Fetch: func(teams []models.Team) ([]*graphql.Team,
                  ↪  []error) {
14                        data := make(map[string]models.DisciplineOffer,
                          ↪  len(teams))
15                        result := make([]*graphql.Team, len(teams))
16                        errors := make([]error, len(teams))
17                        type teamInfo struct {
18                                ID       string
19                                Position int
20                        }
21                        disciplineOfferMap := make(map[string][]teamInfo)
22                        for teamID, team := range teams {
23                                disciplineOfferMap[team.Discipline.ID] =
                                  ↪  append(disciplineOfferMap[team.Discipline.ID],
                                  ↪  teamInfo{
24                                      ID:       team.ID,
25                                      Position: teamID,
26                                })
27                        }
28                        disciplineOfferList := make([]string, 0,
                          ↪  len(disciplineOfferMap))
29                        for key := range disciplineOfferMap {
30                                disciplineOfferList =
                                  ↪  append(disciplineOfferList, key)
31                        }
32                        var queryErrors map[string]error
33                        if err == nil {
34                                queryErrors =
                                  ↪  store.GetMulti(models.TableDisciplineOffers,
                                  ↪  disciplineOfferList, data)
35                        }
```

```
36                      for disciplineOfferID, teamInfos := range
                        ↪  disciplineOfferMap {
37                              row := data[disciplineOfferID]
38                              rowTeams := make(map[string]int,
                                ↪  len(row.Teams))
39                              for i, team := range row.Teams {
40                                      rowTeams[team.ID] = i
41                              }
42                              for _, team := range teamInfos {
43                                      errors[team.Position] = err
44                                      if errors[team.Position] == nil {
45                                              errors[team.Position] =
                                                ↪  queryErrors[disciplineOfferID]
46                                      }
47                                      if teamIndex, ok :=
                                        ↪  rowTeams[team.ID]; ok &&
                                        ↪  errors[team.Position] == nil
                                        ↪  {
48                                              resultTeam :=
                                                ↪  graphql.ToTeam(&row,
                                                ↪  teamIndex)
49                                              result[team.Position] =
                                                ↪  &resultTeam
50                                      }
51                              }
52                      }
53                      return result, errors
54              },
55      })
56 }
```

Arquivo university_loader.go

```
1  package dataloaders
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/util/kv"
6  )
7
```

```
8  func getUniversityLoaderFromStore(store kv.Store, err error, options
   ↪  Options) *UniversityLoader {
9          return NewUniversityLoader(UniversityLoaderConfig{
10                 MaxBatch: options.MaxBatch,
11                 Wait:     options.Wait,
12                 Fetch: func(keys []string) ([]*models.University,
                   ↪  []error) {
13                         data := make(map[string]*models.University,
                           ↪  len(keys))
14                         result := make([]*models.University, len(keys))
15                         errors := make([]error, len(keys))
16                         var queryErrors map[string]error
17                         if err == nil {
18                                 queryErrors =
                                   ↪  store.GetMulti(models.TableUniversity,
                                   ↪  keys, data)
19                         }
20                         for i, key := range keys {
21                                 errors[i] = err
22                                 if errors[i] == nil {
23                                         errors[i] = queryErrors[key]
24                                 }
25                                 if errors[i] == nil {
26                                         result[i] = data[key]
27                                 }
28                         }
29                         return result, errors
30                 },
31         })
32 }
```

Arquivo user_loader.go

```
1  package dataloaders
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/util/kv"
6  )
7
```

```
8  func getUserLoaderFromStore(store kv.Store, err error, options Options)
   ↪  *UserLoader {
9          return NewUserLoader(UserLoaderConfig{
10                 MaxBatch: options.MaxBatch,
11                 Wait:     options.Wait,
12                 Fetch: func(keys []string) ([]*models.User, []error) {
13                         data := make(map[string]*models.User, len(keys))
14                         result := make([]*models.User, len(keys))
15                         errors := make([]error, len(keys))
16                         var queryErrors map[string]error
17                         if err == nil {
18                                 queryErrors =
                                   ↪  store.GetMulti(models.TableUser, keys,
                                   ↪  data)
19                         }
20                         for i, key := range keys {
21                                 errors[i] = err
22                                 if errors[i] == nil {
23                                         errors[i] = queryErrors[key]
24                                 }
25                                 if errors[i] == nil {
26                                         result[i] = data[key]
27                                 }
28                         }
29                         return result, errors
30                 },
31         })
32 }
```

B.1.8 Pasta graphql/resolvers

Arquivo base.go

```
1  package resolvers
2
3  import "errors"
4
5  var errAccessDenied = errors.New("Access Denied")
```

Arquivo campus_resolver.go

```go
package resolvers

import (
        "context"

        "github.com/fjorgemota/gurudamatricula/graphql"
        "github.com/fjorgemota/gurudamatricula/models"
)

type campusResolver struct {
        handler Handler
}

func (r campusResolver) University(ctx context.Context, obj
↪  *models.Campus) (*models.University, error) {
        return loadUniversity(ctx, obj.UniversityID)
}

func (r campusResolver) Period(ctx context.Context, obj *models.Campus)
↪  (*models.Period, error) {
        return loadPeriod(ctx, obj.Period)
}

func (r campusResolver) DisciplineOffers(ctx context.Context, obj
↪  *models.Campus) ([]*models.DisciplineOffer, error) {
        return loadDisciplineOffers(ctx, obj.DisciplineOffers)
}

func (r campusResolver) SearchDisciplineOffers(ctx context.Context, obj
↪  *models.Campus, query *graphql.SearchExpression, options
↪  *graphql.SearchOptions) (*graphql.DisciplineOfferSearchResult, error)
↪  {
        newQuery := getNewQuery(query, "campus_id", obj.ID)
        return searchDisciplineOffers(ctx, r.handler, obj.Period.ID,
        ↪  newQuery, options)
}

```

```
31 func (r campusResolver) SearchTeams(ctx context.Context, obj
   ↪ *models.Campus, query *graphql.SearchExpression, options
   ↪ *graphql.SearchOptions) (*graphql.TeamSearchResult, error) {
32         newQuery := getNewQuery(query, "campus_id", obj.ID)
33         return searchTeams(ctx, r.handler, obj.Period.ID, newQuery,
           ↪ options)
34 }
```

Arquivo collection.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/99designs/gqlgen/graphql"
7  )
8
9  func answerWithSummaryOnly(ctx context.Context, summaryFields []string)
   ↪ bool {
10         var zero struct{}
11         fields := graphql.CollectAllFields(ctx)
12         inSummary := make(map[string]struct{}, 1+len(summaryFields))
13         inSummary["__typename"] = zero
14         for _, field := range summaryFields {
15                 inSummary[field] = zero
16         }
17         for _, field := range fields {
18                 if _, ok := inSummary[field]; !ok {
19                         return false
20                 }
21         }
22         return true
23 }
```

Arquivo course_resolver.go

```
1  package resolvers
2
```

```
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/models"
7  )
8
9  type courseResolver struct {
10         handler Handler
11 }
12
13 func (r courseResolver) University(ctx context.Context, obj
   ↪  *models.Course) (*models.University, error) {
14         return loadUniversity(ctx, obj.UniversityID)
15 }
16
17 func (r courseResolver) Habilitations(ctx context.Context, obj
   ↪  *models.Course) ([]*models.Habilitation, error) {
18         return loadHabilitations(ctx, obj.Habilitations)
19 }
```

Arquivo discipline_offer_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5          "errors"
6
7          "github.com/fjorgemota/gurudamatricula/graphql"
8          "github.com/fjorgemota/gurudamatricula/models"
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10 )
11
12 type courseOfferResolver struct {
13         handler Handler
14 }
15
16 func (r courseOfferResolver) Exclusivity(ctx context.Context, obj
   ↪  *models.DisciplineOfferTeamCourse) (graphql.Exclusivity, error) {
17         switch format.Exclusivity(obj.Exclusivity) {
```

```
18          case format.Exclusive:
19                  return graphql.ExclusivityExclusive, nil
20          case format.NotExclusive:
21                  return graphql.ExclusivityNotExclusive, nil
22          }
23          return graphql.ExclusivityUnknown, nil
24  }
25
26  func (r courseOfferResolver) SearchTeams(ctx context.Context, obj
    ↪  *models.DisciplineOfferTeamCourse, periodID string, query
    ↪  *graphql.SearchExpression, options *graphql.SearchOptions)
    ↪  (*graphql.TeamSearchResult, error) {
27          newQuery := getNewQuery(query, "course_id", obj.ID)
28          return searchTeams(ctx, r.handler, periodID, newQuery, options)
29  }
30
31  type hourMinuteResolver struct{}
32
33  func (r hourMinuteResolver) Hour(ctx context.Context, obj
    ↪  *models.DisciplineOfferTeamScheduleHourMinute) (int, error) {
34          return int(obj.Hour), nil
35  }
36
37  func (r hourMinuteResolver) Minute(ctx context.Context, obj
    ↪  *models.DisciplineOfferTeamScheduleHourMinute) (int, error) {
38          return int(obj.Minute), nil
39  }
40
41  type disciplineOfferResolver struct {
42          handler Handler
43  }
44
45  func (r disciplineOfferResolver) University(ctx context.Context, obj
    ↪  *models.DisciplineOffer) (*models.University, error) {
46          return loadUniversity(ctx, obj.UniversityID)
47  }
48
49  func (r disciplineOfferResolver) Campus(ctx context.Context, obj
    ↪  *models.DisciplineOffer) (*models.Campus, error) {
```

```
50          return loadCampus(ctx, obj.Campus)
51  }
52
53  func (r disciplineOfferResolver) Period(ctx context.Context, obj
    ↪  *models.DisciplineOffer) (*models.Period, error) {
54          return loadPeriod(ctx, obj.Period)
55  }
56
57  func (r disciplineOfferResolver) Teams(ctx context.Context, obj
    ↪  *models.DisciplineOffer) ([]*graphql.Team, error) {
58          teams := make([]*graphql.Team, len(obj.Teams))
59          for i := range teams {
60                  team := graphql.ToTeam(obj, i)
61                  teams[i] = &team
62          }
63          return teams, nil
64  }
65
66  func (r disciplineOfferResolver) Team(ctx context.Context, obj
    ↪  *models.DisciplineOffer, id string) (*graphql.Team, error) {
67          var result *graphql.Team
68          for i, team := range obj.Teams {
69                  if team.ID == id {
70                          team := graphql.ToTeam(obj, i)
71                          result = &team
72                  }
73          }
74          var err error
75          if result == nil {
76                  err = errors.New("Team not found")
77          }
78          return result, err
79  }
80
81  func (r disciplineOfferResolver) SearchTeams(ctx context.Context, obj
    ↪  *models.DisciplineOffer, query *graphql.SearchExpression, options
    ↪  *graphql.SearchOptions) (*graphql.TeamSearchResult, error) {
82          newQuery := getNewQuery(query, "discipline_id", obj.ID)
```

```
83          return searchTeams(ctx, r.handler, obj.Period.ID, newQuery,
            ↪  options)
84  }
```

Arquivo discipline_resolver.go

```
1   package resolvers
2
3   import (
4           "context"
5           "strings"
6
7           "github.com/fjorgemota/gurudamatricula/graphql"
8           "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
9           "github.com/fjorgemota/gurudamatricula/models"
10          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
11          "github.com/fjorgemota/gurudamatricula/robot/importer"
12          "github.com/fjorgemota/gurudamatricula/util/indexer"
13          "github.com/fjorgemota/gurudamatricula/util/kv"
14  )
15
16  type disciplineResolver struct {
17          handler Handler
18  }
19
20  func (r disciplineResolver) University(ctx context.Context, obj
    ↪  *models.Discipline) (*models.University, error) {
21          return loadUniversity(ctx, obj.UniversityID)
22  }
23
24  func (r disciplineResolver) Course(ctx context.Context, obj
    ↪  *models.Discipline) (*models.Course, error) {
25          return loadCourse(ctx, obj.Course)
26  }
27
28  func (r disciplineResolver) Step(ctx context.Context, obj
    ↪  *models.Discipline) (*graphql.Step, error) {
29          return &graphql.Step{
30                  HabilitationID: obj.Habilitation.ID,
31                  ID:             obj.Step.ID,
```

```
32                  Name:           obj.Step.Name,
33          }, nil
34  }
35
36  func (r disciplineResolver) Habilitation(ctx context.Context, obj
    ↪  *models.Discipline) (*models.Habilitation, error) {
37          return loadHabilitation(ctx, obj.Habilitation)
38  }
39
40  func (r disciplineResolver) Related(ctx context.Context, obj
    ↪  *models.Discipline) ([]*graphql.DisciplineRelated, error) {
41          result := make([]*graphql.DisciplineRelated, len(obj.Related))
42          for i, related := range obj.Related {
43                  relatedObj := graphql.DisciplineRelated{
44                          Type:
                            ↪  graphql.DisciplineRelatedType(strings.ToUpper(related.T
45                          Name: related.Name,
46                  }
47                  for _, item := range obj.RelatedItems {
48                          if item.RelatedID == i {
49                                  var description *string
50                                  if item.Description != "" {
51                                          localDescription :=
                                            ↪  item.Description
52                                          description = &localDescription
53                                  }
54                                  relatedObj.Items =
                                    ↪  append(relatedObj.Items,
                                    ↪  &graphql.DisciplineRelatedItem{
55                                          Description: description,
56                                          Type:
                                            ↪  graphql.DisciplineRelatedItemType(strin
57                                          Value:       item.Value,
58                                  })
59                          }
60                  }
61                  result[i] = &relatedObj
62          }
63          return result, nil
```

```
64 }
65
66 func (r disciplineResolver) RelatedDisciplines(ctx context.Context, obj
   ↪  *models.Discipline) ([]*models.Discipline, error) {
67         var err error
68         var result []*models.Discipline
69         var zero struct{}
70         specificIdsMap := make(map[string]struct{})
71         genericIdsMap := make(map[string]struct{})
72         for _, item := range obj.RelatedItems {
73                 switch item.Type {
74                 case "discipline_id":
75                         specificIdsMap[item.Value] = zero
76                 case "discipline_generic_id":
77                         genericIdsMap[item.Value] = zero
78                 }
79         }
80         if len(genericIdsMap) > 0 {
81                 var index indexer.Index
82                 index, err = r.handler.GetIndex(ctx, importer.IndexData{
83                         Target:   ddiffer.TargetDiscipline,
84                         ParentID: obj.UniversityID,
85                 })
86                 if err == nil {
87                         expression := indexer.Or{}
88                         for genericID := range genericIdsMap {
89                                 expression.Operations =
                                ↪  append(expression.Operations,
                                ↪  indexer.FieldTerminal{
90                                         Attribute: "generic_id",
91                                         Value:     genericID,
92                                 })
93                         }
94                         it := index.Search(expression)
95                         for err == nil {
96                                 var ref models.BasicDiscipline
97                                 err = it.Next(&ref)
98                                 if err == nil {
99                                         specificIdsMap[ref.ID] = zero
```

```
100                                     }
101                             }
102                             if indexer.IsDoneError(err) {
103                                     err = nil
104                             }
105                     }
106             }
107             disciplineLoader := dataloaders.GetDisciplineLoader(ctx)
108             if err == nil {
109                     result = make([]*models.Discipline, 0,
                        ↪  len(specificIdsMap))
110                     ids := make([]string, 0, len(specificIdsMap))
111                     for specificID := range specificIdsMap {
112                             ids = append(ids, specificID)
113                     }
114                     rows, errors := disciplineLoader.LoadAll(ids)
115                     for i, row := range rows {
116                             if kv.IsKeyNotFoundError(errors[i]) {
117                                     continue
118                             }
119                             if err == nil {
120                                     err = errors[i]
121                             }
122                             if err == nil && row != nil {
123                                     result = append(result, row)
124                             }
125                     }
126             }
127             return result, err
128     }
129
130     func (r disciplineResolver) SearchTeams(ctx context.Context, obj
        ↪  *models.Discipline, periodID string, query *graphql.SearchExpression,
        ↪  options *graphql.SearchOptions) (*graphql.TeamSearchResult, error) {
131             newQuery := getNewQuery(query, "discipline_generic_id",
                ↪  obj.GenericID)
132             return searchTeams(ctx, r.handler, periodID, newQuery, options)
133     }
134
```

```
135 func (r disciplineResolver) SearchDisciplineOffers(ctx context.Context,
    ↪ obj *models.Discipline, periodID string, query
    ↪ *graphql.SearchExpression, options *graphql.SearchOptions)
    ↪ (*graphql.DisciplineOfferSearchResult, error) {
136         newQuery := getNewQuery(query, "generic_id", obj.GenericID)
137         return searchDisciplineOffers(ctx, r.handler, periodID, newQuery,
            ↪ options)
138 }
139
140 func (r disciplineResolver) Tables(ctx context.Context, obj
    ↪ *models.Discipline) ([]*graphql.Table, error) {
141         return resolveTables(obj.Tables, obj.TableColumns, obj.TableRows)
142 }
```

Arquivo foreign_key.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
7          "github.com/fjorgemota/gurudamatricula/util/kv"
8
9          "github.com/fjorgemota/gurudamatricula/models"
10 )
11
12 func loadUniversity(ctx context.Context, universityID string)
   ↪ (*models.University, error) {
13         var result *models.University
14         var err error
15         if answerWithSummaryOnly(ctx, []string{"id"}) {
16                 result = &models.University{
17                         ID: universityID,
18                 }
19         } else {
20                 loader := dataloaders.GetUniversityLoader(ctx)
21                 result, err = loader.Load(universityID)
22         }
23         return result, err
```

```
24 }
25
26 func loadCourse(ctx context.Context, course models.ForeignKey)
   ↪  (*models.Course, error) {
27         var result *models.Course
28         var err error
29         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
30                 result = &models.Course{
31                         ID:   course.ID,
32                         Name: course.Name,
33                 }
34         } else {
35                 loader := dataloaders.GetCourseLoader(ctx)
36                 result, err = loader.Load(course.ID)
37         }
38         return result, err
39 }
40
41 func loadPeriod(ctx context.Context, period models.ForeignKey)
   ↪  (*models.Period, error) {
42         var result *models.Period
43         var err error
44         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
45                 result = &models.Period{
46                         ID:   period.ID,
47                         Name: period.Name,
48                 }
49         } else {
50                 loader := dataloaders.GetPeriodLoader(ctx)
51                 result, err = loader.Load(period.ID)
52         }
53         return result, err
54 }
55
56 func loadCampus(ctx context.Context, campus models.ForeignKey)
   ↪  (*models.Campus, error) {
57         var result *models.Campus
58         var err error
59         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
```

```
60                  result = &models.Campus{
61                          ID:   campus.ID,
62                          Name: campus.Name,
63                  }
64          } else {
65                  loader := dataloaders.GetCampusLoader(ctx)
66                  result, err = loader.Load(campus.ID)
67          }
68          return result, err
69  }
70
71  func loadCampi(ctx context.Context, campi []models.PeriodCampus)
    ↪  ([]*models.Campus, error) {
72          result := make([]*models.Campus, 0, len(campi))
73          var err error
74          if answerWithSummaryOnly(ctx, []string{"id", "name", "olcode"}) {
75                  for _, campus := range campi {
76                          result = append(result, &models.Campus{
77                                  ID:     campus.ID,
78                                  Name:   campus.Name,
79                                  OLCode: campus.OLCode,
80                          })
81                  }
82          } else {
83                  ids := make([]string, len(campi))
84                  for i, campus := range campi {
85                          ids[i] = campus.ID
86                  }
87                  campusLoader := dataloaders.GetCampusLoader(ctx)
88                  results, errors := campusLoader.LoadAll(ids)
89                  for i := range ids {
90                          if kv.IsKeyNotFoundError(errors[i]) {
91                                  continue
92                          }
93                          if err == nil {
94                                  err = errors[i]
95                          }
96                          if err == nil && results[i] != nil {
97                                  result = append(result, results[i])
```

```
98                          }
99                  }
100         }
101         return result, err
102 }
103
104 func loadHabilitations(ctx context.Context, habilitations
    ↪  []models.ForeignKey) ([]*models.Habilitation, error) {
105         var err error
106         result := make([]*models.Habilitation, 0, len(habilitations))
107         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
108                 for _, habilitation := range habilitations {
109                         result = append(result, &models.Habilitation{
110                                 ID:   habilitation.ID,
111                                 Name: habilitation.Name,
112                         })
113                 }
114         } else {
115                 ids := make([]string, len(habilitations))
116                 for i, plan := range habilitations {
117                         ids[i] = plan.ID
118                 }
119                 loader := dataloaders.GetHabilitationLoader(ctx)
120                 results, errors := loader.LoadAll(ids)
121                 for i := range habilitations {
122                         if kv.IsKeyNotFoundError(errors[i]) {
123                                 continue
124                         }
125                         if err == nil {
126                                 err = errors[i]
127                         }
128                         if err == nil && results[i] != nil {
129                                 result = append(result, results[i])
130                         }
131                 }
132         }
133         return result, err
134 }
135
```

```
136 func loadHabilitation(ctx context.Context, habilitation
    ↪  models.ForeignKey) (*models.Habilitation, error) {
137         var result *models.Habilitation
138         var err error
139         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
140                 result = &models.Habilitation{
141                         ID:   habilitation.ID,
142                         Name: habilitation.Name,
143                 }
144         } else {
145                 loader := dataloaders.GetHabilitationLoader(ctx)
146                 result, err = loader.Load(habilitation.ID)
147         }
148         return result, err
149 }
150
151 func loadPeriods(ctx context.Context, periods []models.ForeignKey)
    ↪  ([]*models.Period, error) {
152         var err error
153         result := make([]*models.Period, 0, len(periods))
154         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
155                 for _, period := range periods {
156                         result = append(result, &models.Period{
157                                 ID:   period.ID,
158                                 Name: period.Name,
159                         })
160                 }
161         } else {
162                 keys := make([]string, len(periods))
163                 for i, period := range periods {
164                         keys[i] = period.ID
165                 }
166                 loader := dataloaders.GetPeriodLoader(ctx)
167                 results, errors := loader.LoadAll(keys)
168                 for i := range keys {
169                         if kv.IsKeyNotFoundError(errors[i]) {
170                                 continue
171                         }
172                         if err == nil {
```

```
173                                     err = errors[i]
174                             }
175                             if err == nil && results[i] != nil {
176                                     result = append(result, results[i])
177                             }
178                     }
179             }
180             return result, err
181     }
182
183     func loadCourses(ctx context.Context, courses []models.ForeignKey)
        ↪  ([]*models.Course, error) {
184             result := make([]*models.Course, 0, len(courses))
185             var err error
186             if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
187                     for _, course := range courses {
188                             result = append(result, &models.Course{
189                                     ID:   course.ID,
190                                     Name: course.Name,
191                             })
192                     }
193             } else {
194                     loader := dataloaders.GetCourseLoader(ctx)
195                     ids := make([]string, len(courses))
196                     for i, course := range courses {
197                             ids[i] = course.ID
198                     }
199                     results, errors := loader.LoadAll(ids)
200                     for i := range ids {
201                             if kv.IsKeyNotFoundError(errors[i]) {
202                                     continue
203                             }
204                             if err == nil {
205                                     err = errors[i]
206                             }
207                             if err == nil {
208                                     result = append(result, results[i])
209                             }
210                     }
```

```
211         }
212         return result, err
213 }
214
215 func loadDisciplineOffers(ctx context.Context, disciplineOffers
    ↪  []models.ForeignKey) ([]*models.DisciplineOffer, error) {
216         var err error
217         result := make([]*models.DisciplineOffer, 0,
            ↪  len(disciplineOffers))
218         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
219                 for _, disciplineOffer := range disciplineOffers {
220                         result = append(result, &models.DisciplineOffer{
221                                 ID:   disciplineOffer.ID,
222                                 Name: disciplineOffer.Name,
223                         })
224                 }
225         } else {
226                 ids := make([]string, len(disciplineOffers))
227                 for i, disciplineOffer := range disciplineOffers {
228                         ids[i] = disciplineOffer.ID
229                 }
230                 loader := dataloaders.GetDisciplineOfferLoader(ctx)
231                 results, errors := loader.LoadAll(ids)
232                 for i := range disciplineOffers {
233                         if kv.IsKeyNotFoundError(errors[i]) {
234                                 continue
235                         }
236                         if err == nil {
237                                 err = errors[i]
238                         }
239                         if err == nil && results[i] != nil {
240                                 result = append(result, results[i])
241                         }
242                 }
243         }
244         return result, err
245 }
```

Arquivo habilitation_resolver.go

```go
package resolvers

import (
        "context"

        "github.com/fjorgemota/gurudamatricula/graphql"
        "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
        "github.com/fjorgemota/gurudamatricula/models"
        "github.com/fjorgemota/gurudamatricula/util/kv"
)

type habilitationResolver struct {
        handler Handler
}

func (r habilitationResolver) University(ctx context.Context, obj
↪  *models.Habilitation) (*models.University, error) {
        return loadUniversity(ctx, obj.UniversityID)
}

func (r habilitationResolver) Course(ctx context.Context, obj
↪  *models.Habilitation) (*models.Course, error) {
        return loadCourse(ctx, obj.Course)
}

func (r habilitationResolver) Limits(ctx context.Context, obj
↪  *models.Habilitation) ([]*graphql.HabilitationLimit, error) {
        result := make([]*graphql.HabilitationLimit,
        ↪  len(obj.LimitValues))
        for limitID, objLimit := range obj.Limits {
                limit := graphql.HabilitationLimit{
                        ID:      objLimit.ID,
                        Name:    objLimit.Name,
                        Context: objLimit.Context,
                        Unit:    objLimit.Unit,
                }
                for _, value := range obj.LimitValues {
                        if value.ID == limitID {
```

```
35                                  limit.Values = append(limit.Values,
                                    ↪  &graphql.HabilitationLimitValue{
36                                          ID:    limitID,
37                                          Key:   value.Key,
38                                          Value: value.Value,
39                                  })
40                          }
41                  }
42                  result[limitID] = &limit
43          }
44          return result, nil
45  }
46
47  func (r habilitationResolver) Steps(ctx context.Context, obj
    ↪  *models.Habilitation) ([]*graphql.Step, error) {
48          result := make([]*graphql.Step, len(obj.Steps))
49          for i, objStep := range obj.Steps {
50                  step := graphql.Step{
51                          ID:   objStep.ID,
52                          Name: objStep.Name,
53                  }
54                  for _, objDiscipline := range obj.StepDisciplines {
55                          if objDiscipline.StepID == i {
56                                  step.Disciplines =
                                    ↪  append(step.Disciplines,
                                    ↪  models.ForeignKey{
57                                          ID:
                                            ↪  objDiscipline.Discipline.ID,
58                                          Name:
                                            ↪  objDiscipline.Discipline.Name,
59                                  })
60                          }
61                  }
62                  result[i] = &step
63          }
64          return result, nil
65  }
66
```

```
67 func (r habilitationResolver) Tables(ctx context.Context, obj
   ↪ *models.Habilitation) ([]*graphql.Table, error) {
68         return resolveTables(obj.Tables, obj.TableColumns, obj.TableRows)
69 }
70
71 type stepResolver struct {
72         handler Handler
73 }
74
75 func (r stepResolver) Disciplines(ctx context.Context, obj *graphql.Step)
   ↪ ([]*models.Discipline, error) {
76         var result []*models.Discipline
77         var err error
78         onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name"})
79         disciplines := obj.Disciplines
80         if len(obj.HabilitationID) > 0 {
81                 loader := dataloaders.GetHabilitationLoader(ctx)
82                 var habilitation *models.Habilitation
83                 habilitation, err = loader.Load(obj.HabilitationID)
84                 if err == nil {
85                         disciplines = nil
86                         for i, objStep := range habilitation.Steps {
87                                 if objStep.ID != obj.ID && objStep.Name
                                   ↪ != obj.Name {
88                                         continue
89                                 }
90                                 for _, objDiscipline := range
                                   ↪ habilitation.StepDisciplines {
91                                         if objDiscipline.StepID == i {
92                                                 disciplines =
                                                   ↪ append(disciplines,
                                                   ↪ models.ForeignKey{
93                                                         ID:
                                                           ↪ objDiscipline.Disciplin
94                                                         Name:
                                                           ↪ objDiscipline.Disciplin
95                                                 })
96                                         }
97                                 }
```

```
98                              break
99                      }
100             }
101     }
102     if onlySummary {
103             result = make([]*models.Discipline, len(disciplines))
104             for i, discipline := range disciplines {
105                     result[i] = &models.Discipline{
106                             ID:   discipline.ID,
107                             Name: discipline.Name,
108                     }
109             }
110     } else {
111             ids := make([]string, len(disciplines))
112             for i, discipline := range disciplines {
113                     ids[i] = discipline.ID
114             }
115             loader := dataloaders.GetDisciplineLoader(ctx)
116             results, errors := loader.LoadAll(ids)
117             for i := range ids {
118                     if kv.IsKeyNotFoundError(errors[i]) {
119                             continue
120                     }
121                     if err == nil {
122                             err = errors[i]
123                     }
124                     if err == nil && results[i] != nil {
125                             result = append(result, results[i])
126                     }
127             }
128     }
129     return result, err
130 }
```

Arquivo index.go

```
1  package resolvers
2
3  func getIndex(max int, index *int, end bool) int {
4          var result int
```

```
5           if index == nil {
6                   if end {
7                           result = max
8                   }
9           } else {
10                  result = *index
11          }
12          if result < 0 {
13                  result += max
14          }
15          if result < 0 {
16                  result = 0
17          }
18          if result > max {
19                  result = max
20          }
21          return result
22  }
23
24  func getStartEndIndex(max int, start, end *int) (startResult, endResult
    ↪  int) {
25          startResult = getIndex(max, start, false)
26          endResult = getIndex(max, end, true)
27          return startResult, endResult
28  }
```

Arquivo mutation_resolver.go

```
1   package resolvers
2
3   import (
4           "context"
5           "errors"
6           "strings"
7
8           "github.com/fjorgemota/gurudamatricula/models/keygen"
9           "github.com/fjorgemota/gurudamatricula/robot/fetcher"
10          "github.com/fjorgemota/gurudamatricula/robot/format"
11
12          "github.com/volatiletech/authboss"
```

```
13
14          "github.com/fjorgemota/gurudamatricula/graphql"
15          "github.com/fjorgemota/gurudamatricula/models"
16          "github.com/fjorgemota/gurudamatricula/util/kv"
17  )
18
19  type idVersion struct {
20          ID      string
21          Version string
22  }
23  type mutationResolver struct {
24          handler Handler
25  }
26
27  func (r mutationResolver) SetPlan(ctx context.Context, input
    ↪  graphql.PlanInput) (*models.Plan, error) {
28          var plan *models.Plan
29          var userRef models.ForeignKey
30          var store kv.Store
31          clk := r.handler.GetClock()
32          auth := r.handler.GetAuthBoss()
33          req := r.handler.GetRequest(ctx)
34          userInstance, err := auth.CurrentUser(req)
35          if err == authboss.ErrUserNotFound || err == nil && userInstance
            ↪  == nil {
36                  err = errAccessDenied
37          }
38          if err == nil {
39                  userModel := userInstance.(*models.User)
40                  userRef.ID, err = keygen.GenerateUserID(userModel.ID)
41                  userRef.Name = userModel.Name
42          }
43          if err == nil && len(input.Possibilities) == 0 {
44                  err = errors.New("Cannot create plan without any
                    ↪  possibilities")
45          }
46          if err == nil {
47                  store, err = r.handler.GetStore(ctx)
48          }
```

```
49          var newVersion models.PlanVersion
50          if err == nil {
51                  newVersion.ID, err =
                    ↪  store.GenerateID(models.TablePlanVersion)
52                  newVersion.SavedAt = clk.Now()
53                  newVersion.SelectedPossibility =
                    ↪  input.SelectedPossibility
54                  type disciplineOffer struct {
55                          ID      string
56                          Version string
57                  }
58                  schedulesIndex := make(map[graphql.PlanScheduleInput]int)
59                  teachersIndex := make(map[string]int)
60                  roomsIndex := make(map[string]int)
61                  offersIndex := make(map[idVersion]int)
62                  disciplinesIndex := make(map[idVersion]int)
63                  teamsIndex := make(map[string]int)
64                  newVersion.Disciplines =
                    ↪  make([]models.PlanVersionDiscipline,
                    ↪  len(input.Disciplines))
65                  for index, discipline := range input.Disciplines {
66                          disciplineIDVersion := idVersion{
67                                  ID:      discipline.ID,
68                                  Version: discipline.Version,
69                          }
70                          if _, ok := disciplinesIndex[disciplineIDVersion];
                            ↪  ok && err == nil {
71                                  err = errors.New("Cannot put duplicated
                                    ↪  discipline id/version")
72                          }
73                          disciplinesIndex[disciplineIDVersion] = index
74                          newVersion.Disciplines[index] =
                            ↪  models.PlanVersionDiscipline{
75                                  ID:           discipline.ID,
76                                  GenericID:    discipline.GenericID,
77                                  Code:         discipline.Code,
78                                  Name:         discipline.Name,
79                                  Type:         discipline.Type,
80                                  Description:  discipline.Description,
```

```
81                          Version:      discipline.Version,
82                          Course:       *discipline.Course,
83                          Habilitation: *discipline.Habilitation,
84                          Step:         *discipline.Step,
85                  }
86                  for _, item := range
                    ↪ discipline.RelatedDisciplines {
87                          newVersion.DisciplinesRelatedDisciplines
                            ↪ =
                            ↪ append(newVersion.DisciplinesRelatedDisciplines,
                            ↪ models.PlanVersionRelatedDisciplineSummary{
88                              DisciplineID: index,
89                              ID:           item.ID,
90                              GenericID:    item.GenericID,
91                              Version:      item.Version,
92                              Code:         item.Code,
93                              Name:         item.Name,
94                              Type:         item.Type,
95                              Description:  item.Description,
96                              Course:       *item.Course,
97                              Habilitation: *item.Habilitation,
98                              Step:         *item.Step,
99                          })
100                 }
101                 for _, related := range discipline.Related {
102                         relatedIndex :=
                            ↪ len(newVersion.DisciplinesRelated)
103                         newVersion.DisciplinesRelated =
                            ↪ append(newVersion.DisciplinesRelated,
                            ↪ models.PlanVersionDisciplineRelated{
104                             DisciplineID: index,
105                             Name:         related.Name,
106                             Type:         related.Type,
107                         })
108                         for _, item := range related.Items {
109                             relatedItem :=
                                ↪ models.DisciplineRelatedItem{
110                                 RelatedID: relatedIndex,
111                                 Type:      item.Type,
```

```
112                                             Value:     item.Value,
113                                     }
114                                     if item.Description != nil {
115                                             relatedItem.Description =
                                                ↪  *item.Description
116                                     }
117                                     newVersion.DisciplinesRelatedItem
                                        ↪  =
                                        ↪  append(newVersion.DisciplinesRelatedIte
                                        ↪  relatedItem)
118                             }
119                     }
120                     if err == nil {
121                             err = r.addTables(&newVersion,
                                ↪  discipline.Tables, index,
                                ↪  models.DisciplineTableType)
122                     }
123             }
124             newVersion.DisciplinesOffers =
            ↪  make([]models.PlanVersionDisciplineOffer,
            ↪  len(input.Offers))
125             for index, offer := range input.Offers {
126                     offerIDVersion := idVersion{
127                             ID:      offer.ID,
128                             Version: offer.Version,
129                     }
130                     if _, ok := offersIndex[offerIDVersion]; ok &&
                        ↪  err == nil {
131                             err = errors.New("Cannot put duplicated
                                ↪  offer id/version")
132                     }
133                     offersIndex[offerIDVersion] = index
134                     newVersion.DisciplinesOffers[index] =
                        ↪  models.PlanVersionDisciplineOffer{
135                             ID:      offer.ID,
136                             Code:    offer.Code,
137                             Custom:  offer.Custom,
138                             Name:    offer.Name,
139                             Version: offer.Version,
```

```
140                     }
141                     if offer.GenericID != nil {
142                             newVersion.DisciplinesOffers[index].GenericID
                                ↪  = *offer.GenericID
143                     }
144                     if offer.Campus != nil {
145                             newVersion.DisciplinesOffers[index].Campus
                                ↪  = *offer.Campus
146                     }
147                     for _, team := range offer.Teams {
148                             teamIndex := len(newVersion.Teams)
149                             teamsIndex[team.ID] = teamIndex
150                             var course
                                ↪  models.DisciplineOfferTeamCourse
151                             if team.Course != nil {
152                                     var exclusivity
                                        ↪  format.Exclusivity
153                                     switch team.Course.Exclusivity {
154                                     case
                                        ↪  graphql.ExclusivityExclusive:
155                                             exclusivity =
                                                ↪  format.Exclusive
156                                     case
                                        ↪  graphql.ExclusivityNotExclusive:
157                                             exclusivity =
                                                ↪  format.NotExclusive
158                                     case graphql.ExclusivityUnknown:
159                                             exclusivity =
                                                ↪  format.Unknown
160                                     }
161                                     course =
                                        ↪  models.DisciplineOfferTeamCourse{
162                                             ID:
                                                ↪  team.Course.ID,
163                                             Name:
                                                ↪  team.Course.Name,
164                                             Exclusivity:
                                                ↪  string(exclusivity),
165                                     }
```

```
166                     }
167                     var vacancies
                        ↪  models.DisciplineOfferTeamVacancy
168                     if team.Vacancies != nil {
169                             vacancies = *team.Vacancies
170                     }
171                     newVersion.Teams =
                        ↪  append(newVersion.Teams,
                        ↪  models.PlanVersionTeam{
172                             ID:                 team.ID,
173                             Code:               team.Code,
174                             Course:             course,
175                             DisciplineOfferID: index,
176                             Vacancies:          vacancies,
177                             Version:            team.Version,
178                     })
179                     for _, teacher := range team.Teachers {
180                             if _, ok :=
                                ↪  teachersIndex[teacher.ID];
                                ↪  !ok {
181                                     newTeacher :=
                                        ↪  models.DisciplineOfferTeacher{
182                                             ID:
                                                ↪  teacher.ID,
183                                             GenericID:
                                                ↪  teacher.GenericID,
184                                             Name:
                                                ↪  teacher.Name,
185                                     }
186                                     if teacher.URL != nil {
187                                             newTeacher.URL =
                                                ↪  *teacher.URL
188                                     }
189                                     teachersIndex[teacher.ID]
                                        ↪  =
                                        ↪  len(newVersion.Teachers)
190                                     newVersion.Teachers =
                                        ↪  append(newVersion.Teachers,
                                        ↪  newTeacher)
```

```
191                             }
192                             newVersion.TeamTeachers =
                                ↪ append(newVersion.TeamTeachers,
                                ↪ models.DisciplineOfferTeamTeacher{
193                                     TeamID:    teamIndex,
194                                     TeacherID:
                                        ↪ teachersIndex[teacher.ID],
195                             })
196                     }
197                     for _, schedule := range team.Schedules {
198                             if _, ok :=
                                ↪ schedulesIndex[*schedule];
                                ↪ !ok {
199                                     newSchedule :=
                                        ↪ models.DisciplineOfferSchedule{
200                                             Start:
                                                ↪ models.DisciplineOfferTeamSc
201                                                     Hour:
                                                        ↪ int8(schedule.Start.
202                                                     Minute:
                                                        ↪ int8(schedule.Start.
203                                             },
204                                             End:
                                                ↪ models.DisciplineOfferTeamSc
205                                                     Hour:
                                                        ↪ int8(schedule.End.Ho
206                                                     Minute:
                                                        ↪ int8(schedule.End.Mi
207                                             },
208                                             DayOfWeek:
                                                ↪ int8(schedule.DayOfWeek),
209                                     }
210                                     schedulesIndex[*schedule]
                                        ↪ =
                                        ↪ len(newVersion.Schedules)
211                                     newVersion.Schedules =
                                        ↪ append(newVersion.Schedules,
                                        ↪ newSchedule)
212                             }
```

```
213                                 roomIndex := -1
214                                 if schedule.Room != nil {
215                                         if _, ok :=
                                            ↪  roomsIndex[schedule.Room.ID];
                                            ↪  !ok {
216                                                 newRoom :=
                                                    ↪  models.DisciplineOfferR
217                                                         ID:
                                                            ↪  schedule.Room.I
218                                                         GenericID:
                                                            ↪  schedule.Room.G
219                                                         Name:
                                                            ↪  schedule.Room.N
220                                                 }
221                                                 if
                                                    ↪  schedule.Room.Olcode
                                                    ↪  != nil {
222                                                         newRoom.OLCode
                                                            ↪  =
                                                            ↪  *schedule.Room.
223                                                 }
224                                                 if
                                                    ↪  schedule.Room.Descripti
                                                    ↪  != nil {
225                                                         newRoom.Description
                                                            ↪  =
                                                            ↪  *schedule.Room.
226                                                 }
227                                                 roomsIndex[newRoom.ID]
                                                    ↪  =
                                                    ↪  len(newVersion.Rooms)
228                                                 newVersion.Rooms
                                                    ↪  =
                                                    ↪  append(newVersion.Rooms
                                                    ↪  newRoom)
229                                         }
230                                         roomIndex =
                                            ↪  roomsIndex[schedule.Room.ID]
231                                 }
```

```
232                                     newVersion.TeamSchedules =
                                        ↪ append(newVersion.TeamSchedules,
                                        ↪ models.DisciplineOfferTeamSchedule{
233                                             TeamID:     teamIndex,
234                                             ScheduleID:
                                                ↪ schedulesIndex[*schedule],
235                                             RoomID:     roomIndex,
236                                     })
237                             }
238                             if err == nil {
239                                     err = r.addTables(&newVersion,
                                        ↪ team.Tables, teamIndex,
                                        ↪ models.TeamTableType)
240                             }
241                     }
242             }
243             newVersion.TotalPossibilities = len(input.Possibilities)
244             newVersion.Possibilities = make([]models.PlanPossibility,
                ↪ len(input.Possibilities))
245             for i, possibility := range input.Possibilities {
246                     newVersion.Possibilities[i].Name =
                        ↪ possibility.Name
247                     for _, selection := range
                        ↪ possibility.SelectionTeams {
248                             offerIDVersion := idVersion{
249                                     ID:      selection.OfferID,
250                                     Version: selection.OfferVersion,
251                             }
252                             if _, ok := offersIndex[offerIDVersion];
                                ↪ !ok && err == nil {
253                                     err = errors.New("Cannot add
                                        ↪ offer id/version that doesn't
                                        ↪ exist to possibility")
254                             }
255                             newVersion.PossibilitiesTeams =
                                ↪ append(newVersion.PossibilitiesTeams,
                                ↪ models.PlanPossibilityTeam{
256                                     Enabled:       selection.Enabled,
```

```
257                                     Selected:
                                        ↪ selection.Selected,
258                                     TeamID:
                                        ↪ teamsIndex[selection.TeamID],
259                                     Color:          selection.Color,
260                                     OfferID:
                                        ↪ offersIndex[offerIDVersion],
261                                     PossibilityID: i,
262                             })
263                     }
264                     for _, selection := range
                        ↪ possibility.SelectionDisciplines {
265                             disciplineIDVersion := idVersion{
266                                     ID:      selection.DisciplineID,
267                                     Version:
                                        ↪ selection.DisciplineVersion,
268                             }
269                             if _, ok :=
                                ↪ disciplinesIndex[disciplineIDVersion];
                                ↪ !ok && err == nil {
270                                     err = errors.New("Cannot add
                                        ↪ discipline id/version that
                                        ↪ doesn't exist to
                                        ↪ possibility")
271                             }
272                             newVersion.PossibilitiesDisciplines =
                                ↪ append(newVersion.PossibilitiesDisciplines,
                                ↪ models.PlanPossibilityDiscipline{
273                                     DisciplineID:
                                        ↪ disciplinesIndex[disciplineIDVersion],
274                                     Enabled:       selection.Enabled,
275                                     PossibilityID: i,
276                             })
277                     }
278             }
279     }
280     var created bool
281     var id string
282     if err == nil {
```

```
283                 if input.ID == nil {
284                         created = true
285                         id, err = store.GenerateID(models.TablePlan)
286                 } else {
287                         id = *input.ID
288                 }
289         }
290         if err == nil {
291                 refs := []kv.Reference{
292                         {Table: models.TableUser, Key: userRef.ID},
293                         {Table: models.TablePlan, Key: id},
294                         {Table: models.TablePlanVersion, Key:
                        ↪  newVersion.ID},
295                         {Table: models.TableUniversity, Key:
                        ↪  input.UniversityID},
296                 }
297                 var versionsToDelete []string
298                 err = store.Transaction(refs, func(db kv.LimitedStore)
                ↪  error {
299                         plan = &models.Plan{}
300                         localErr := db.Get(models.TablePlan, id, plan)
301                         if kv.IsKeyNotFoundError(localErr) && created {
302                                 localErr = nil
303                                 plan = &models.Plan{
304                                         ID:         id,
305                                         User:       userRef,
306                                         University: models.ForeignKey{ID:
                                        ↪  input.UniversityID},
307                                         Period:     models.ForeignKey{ID:
                                        ↪  input.PeriodID},
308                                         CreatedAt:  newVersion.SavedAt,
309                                 }
310                         }
311                         if localErr == nil {
312                                 authorized := plan.User.ID == userRef.ID
313                                 authorized = authorized &&
                                ↪  plan.University.ID ==
                                ↪  input.UniversityID
```

```
314                     authorized = authorized && plan.Period.ID
                        ↪  == input.PeriodID
315                     if !authorized {
316                             localErr = errors.New("Invalid
                                ↪  operation")
317                     }
318             }
319             if localErr == nil {
320                     localErr = db.Set(models.TablePlanVersion,
                        ↪  newVersion.ID, &newVersion)
321             }
322             var universityForeignKey, periodForeignKey
                ↪  models.ForeignKey
323             if localErr == nil {
324                     var university models.University
325                     localErr = db.Get(models.TableUniversity,
                        ↪  input.UniversityID, &university)
326                     if localErr == nil {
327                             universityForeignKey =
                                ↪  models.ForeignKey{
328                                     ID:   input.UniversityID,
329                                     Name: university.Name,
330                             }
331                             for _, period := range
                                ↪  university.Periods {
332                                     if period.ID ==
                                        ↪  input.PeriodID {
333                                             periodForeignKey
                                                ↪  = period
334                                             break
335                                     }
336                             }
337                             if periodForeignKey.ID !=
                                ↪  input.PeriodID ||
                                ↪  periodForeignKey.ID == "" {
338                                     localErr =
                                        ↪  errors.New("Period
                                        ↪  not found")
339                             }
```

```
340                         }
341                 }
342                 if localErr == nil {
343                         newPlan := models.Plan{
344                                 ID:           id,
345                                 User:         userRef,
346                                 University:
                                    ↪ universityForeignKey,
347                                 Period:       periodForeignKey,
348                                 Public:       input.Public,
349                                 Name:         input.Name,
350                                 CreatedAt:    plan.CreatedAt,
351                                 LastModified: newVersion.SavedAt,
352                         }
353                         newPlan.Versions =
                            ↪ append(newPlan.Versions,
                            ↪ models.PlanForeignVersion{
354                                 ID:
                                    ↪ newVersion.ID,
355                                 SavedAt:
                                    ↪ newVersion.SavedAt,
356                                 TotalPossibilities:
                                    ↪ len(newVersion.Possibilities),
357                         })
358                         newPlan.Versions =
                            ↪ append(newPlan.Versions,
                            ↪ plan.Versions...)
359                         if len(newPlan.Versions) > 10 {
360                                 for _, version :=
                                    ↪ range
                                    ↪ newPlan.Versions[10:] {
361                                         versionsToDelete =
                                            ↪ append(versionsToDelete,
                                            ↪ version.ID)
362                                 }
363                                 newPlan.Versions =
                                    ↪ newPlan.Versions[:10]
364                         }
365                         versionsIndexes := make(map[string]int,
                            ↪ len(newPlan.Versions))
```

```
366                     for index, version := range
                        ↪  newPlan.Versions {
367                         versionsIndexes[version.ID] =
                            ↪  index
368                     }
369                     newSchedules :=
                        ↪  make(map[models.PlanSchedulePreview]int,
                        ↪  len(plan.Schedules))
370                     for _, versionSchedule := range
                        ↪  plan.VersionSchedules {
371                         oldVersion :=
                            ↪  plan.Versions[versionSchedule.Version]
372                         if newVersionIndex, ok :=
                            ↪  versionsIndexes[oldVersion.ID];
                            ↪  ok {
373                             schedule :=
                                ↪  plan.Schedules[versionSchedule.
374                             if _, ok :=
                                ↪  newSchedules[schedule];
                                ↪  !ok {
375                                 newSchedules[schedule]
                                    ↪  =
                                    ↪  len(newSchedules)
376                             }
377                             newPlan.VersionSchedules
                                ↪  =
                                ↪  append(newPlan.VersionSchedules
                                ↪  models.PlanVersionSchedule{
378                                 Schedule:
                                    ↪  newSchedules[schedule],
379                                 Version:
                                    ↪  newVersionIndex,
380                             })
381                         }
382                     }
383                     for _, possibilityTeam := range
                        ↪  newVersion.PossibilitiesTeams {
```

```
384                                 if possibilityTeam.PossibilityID
                                    ↪  ==
                                    ↪  newVersion.SelectedPossibility
                                    ↪  && possibilityTeam.Enabled &&
                                    ↪  possibilityTeam.Selected {
385                                         for _, teamSchedule :=
                                            ↪  range
                                            ↪  newVersion.TeamSchedules
                                            ↪  {
386                                                 if
                                                    ↪  teamSchedule.TeamID
                                                    ↪  ==
                                                    ↪  possibilityTeam.TeamID
                                                    ↪  {
387                                                         schedule
                                                            ↪  :=
                                                            ↪  newVersion.Schedules
388                                                         team :=
                                                            ↪  newVersion.Teams[tea
389                                                         disciplineOffer
                                                            ↪  :=
                                                            ↪  newVersion.Disciplin
390                                                         newSchedule
                                                            ↪  :=
                                                            ↪  models.PlanScheduleP
391                                                             Interval:
                                                                ↪  schedule,
392                                                             ID:
                                                                ↪  disciplineOf
393                                                             Title:
                                                                ↪  disciplineOf
394                                                             Color:
                                                                ↪  possibilityT
395                                                         }
396                                                         if _, ok
                                                            ↪  :=
                                                            ↪  newSchedules[newSche
                                                            ↪  !ok {
```

```
397                                                                 newSchedule
                                                                    ↪ =
                                                                    ↪ len(new
398                                                             }
399                                                             newPlan.VersionSche
                                                                ↪ =
                                                                ↪ append(newPlan.
                                                                ↪ models.PlanVers
400                                                                 Schedule:
                                                                    ↪ newSche
401                                                                 Version:
                                                                    ↪ version
402                                                             })
403                                                     }
404                                                 }
405                                             }
406                                     }
407                                     newPlan.Schedules =
                                        ↪ make([]models.PlanSchedulePreview,
                                        ↪ len(newSchedules))
408                                     for schedule, index := range newSchedules
                                        ↪ {
409                                             newPlan.Schedules[index] =
                                                ↪ schedule
410                                     }
411                                     localErr = db.Set(models.TablePlan, id,
                                        ↪ &newPlan)
412                                     plan = &newPlan
413                             }
414                             var user models.User
415                             if localErr == nil {
416                                     localErr = db.Get(models.TableUser,
                                        ↪ userRef.ID, &user)
417                             }
418                             if localErr == nil {
419                                     userPlan := models.UserPlan{
420                                             ID:           plan.ID,
421                                             Name:         plan.Name,
422                                             CreatedAt:    plan.CreatedAt,
```

```
423                                     LastModified: plan.LastModified,
424                                     Period:       plan.Period,
425                                     University:   plan.University,
426                             }
427                             if created {
428                                     user.Plans = append(user.Plans,
                                        ↪ userPlan)
429                             } else {
430                                     for i := range user.Plans {
431                                             if user.Plans[i].ID ==
                                                ↪ plan.ID {
432                                                     user.Plans[i] =
                                                        ↪ userPlan
433                                             }
434                                     }
435                             }
436                             localErr = db.Set(models.TableUser,
                                ↪ userRef.ID, &user)
437                     }
438                     return localErr
439             })
440             if err == nil {
441                     err = r.deletePlanVersions(store,
                        ↪ versionsToDelete)
442             }
443     }
444     return plan, err
445 }
446
447 func (r mutationResolver) addTables(newVersion *models.PlanVersion,
    ↪ tables []*graphql.PlanTableInput, index int, tableType
    ↪ models.PlanTableType) error {
448     var err error
449     for _, table := range tables {
450             newTable := models.PlanTable{
451                     Type:  tableType,
452                     Index: index,
453                     ID:    table.ID,
454                     Title: table.Title,
```

```
455                 }
456                 if table.Description != nil {
457                         newTable.Description = *table.Description
458                 }
459                 tableIndex := len(newVersion.Tables)
460                 newVersion.Tables = append(newVersion.Tables, newTable)
461                 startColumns := len(newVersion.TableColumns)
462                 for _, column := range table.Columns {
463                         newColumn := models.TableColumn{
464                                 ID:      column.ID,
465                                 Name:    column.Name,
466                                 TableID: tableIndex,
467                         }
468                         newVersion.TableColumns =
                        ↪  append(newVersion.TableColumns, newColumn)
469                 }
470                 for _, row := range table.Rows {
471                         if err == nil && (row.Column < 0 || row.Column >=
                        ↪  len(table.Columns)) {
472                                 err = errors.New("Invalid column")
473                         }
474                         newRow := models.TableRow{
475                                 Row:    row.Row,
476                                 Column: startColumns + row.Column,
477                                 Value:  row.Value,
478                         }
479                         newVersion.TableRows =
                        ↪  append(newVersion.TableRows, newRow)
480                 }
481         }
482         return err
483 }
484
485 func (r mutationResolver) deletePlanVersions(store kv.Store, ids
    ↪  []string) error {
486         var err error
487         for _, id := range ids {
488                 if err == nil {
489                         err = store.Del(models.TablePlanVersion, id)
```

```
490                 }
491         }
492         return err
493 }
494
495 func (r mutationResolver) DeletePlan(ctx context.Context, id string)
    ↪  (bool, error) {
496         var store kv.Store
497         auth := r.handler.GetAuthBoss()
498         req := r.handler.GetRequest(ctx)
499         userKey, err := auth.CurrentUserID(req)
500         if err == authboss.ErrUserNotFound || err == nil && len(userKey)
            ↪  == 0 {
501                 err = errAccessDenied
502         }
503         if err == nil {
504                 store, err = r.handler.GetStore(ctx)
505         }
506         if err == nil {
507                 userKey, err = keygen.GenerateUserID(userKey)
508         }
509         if err == nil {
510                 refs := []kv.Reference{
511                         {Table: models.TablePlan, Key: id},
512                         {Table: models.TableUser, Key: userKey},
513                 }
514                 var versionsToDelete []string
515                 err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪  error {
516                         var plan models.Plan
517                         localErr := db.Get(models.TablePlan, id, &plan)
518                         if localErr == nil && plan.User.ID != userKey {
519                                 localErr = errAccessDenied
520                         }
521                         if localErr == nil {
522                                 for _, version := range plan.Versions {
523                                         versionsToDelete =
                                            ↪  append(versionsToDelete,
                                            ↪  version.ID)
```

```
524                                 }
525                                 localErr = db.Del(models.TablePlan, id)
526                         }
527                         var user models.User
528                         if localErr == nil {
529                                 localErr = db.Get(models.TableUser,
                                    ↪  userKey, &user)
530                         }
531                         if localErr == nil {
532                                 userPlans := user.Plans[:0]
533                                 for _, userPlan := range user.Plans {
534                                         if userPlan.ID != id {
535                                                 userPlans =
                                                    ↪  append(userPlans,
                                                    ↪  userPlan)
536                                         }
537                                 }
538                                 user.Plans = userPlans
539                                 localErr = db.Set(models.TableUser,
                                    ↪  userKey, &user)
540                         }
541                         return localErr
542                 })
543                 if err == nil {
544                         err = r.deletePlanVersions(store,
                            ↪  versionsToDelete)
545                 }
546         }
547         return err == nil, err
548 }
549
550 func (r mutationResolver) SetUniversity(ctx context.Context, input
    ↪  graphql.UniversityInput) (*models.University, error) {
551         var store kv.Store
552         var result *models.University
553         var userRef models.ForeignKey
554         auth := r.handler.GetAuthBoss()
555         req := r.handler.GetRequest(ctx)
556         userInstance, err := auth.CurrentUser(req)
```

```
557         if err == authboss.ErrUserNotFound || err == nil && userInstance
            ↪  == nil {
558                 err = errAccessDenied
559         }
560         if err == nil {
561                 userModel := userInstance.(*models.User)
562                 userRef.ID, err = keygen.GenerateUserID(userModel.ID)
563                 userRef.Name = userModel.Name
564         }
565         if err == nil && (!strings.Contains(input.Habilitations.BaseURL,
            ↪  "{filename}") || !strings.Contains(input.Offers.BaseURL,
            ↪  "{filename}")) {
566                 err = errors.New("Base URL should contain {filename}")
567         }
568         if err == nil {
569                 store, err = r.handler.GetStore(ctx)
570         }
571         var created bool
572         var id string
573         if err == nil && input.ID == nil {
574                 created = true
575                 id, err = store.GenerateID(models.TableUniversity)
576         } else if err == nil {
577                 id = *input.ID
578         }
579         if err == nil {
580                 refs := []kv.Reference{
581                         {Table: models.TableUniversity, Key: id},
582                         {Table: models.TableUser, Key: userRef.ID},
583                 }
584                 err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪  error {
585                         var university models.University
586                         localErr := db.Get(models.TableUniversity, id,
                            ↪  &university)
587                         if kv.IsKeyNotFoundError(localErr) && created {
588                                 localErr = nil
589                                 university = models.University{
590                                         ID:   id,
```

```
591                             User: userRef,
592                     }
593             }
594             if localErr == nil && university.User.ID !=
                ↪  userRef.ID {
595                     localErr = errAccessDenied
596             }
597             var secretHabilitations, secretOffers
                ↪  fetcher.Secret
598             if localErr == nil && (created ||
                ↪  input.Habilitations.GenerateSecret) {
599                     secretHabilitations, localErr =
                        ↪  fetcher.GenerateSecret()
600             }
601             if localErr == nil && (created ||
                ↪  input.Offers.GenerateSecret) {
602                     secretOffers, localErr =
                        ↪  fetcher.GenerateSecret()
603             }
604             updateUser := created || input.Name !=
                ↪  university.Name ||
605                     input.Acronym != university.Acronym ||
                        ↪  input.Public != university.Public
606             if localErr == nil {
607                     university.Acronym = input.Acronym
608                     university.Name = input.Name
609                     university.Public = input.Public
610                     university.Habilitations.UserID =
                        ↪  university.User.ID
611                     university.Habilitations.AboutURL =
                        ↪  input.Habilitations.AboutURL
612                     university.Habilitations.BaseURL =
                        ↪  input.Habilitations.BaseURL
613                     university.Habilitations.DeleteURL =
                        ↪  input.Habilitations.DeleteURL
614                     university.Offers.UserID =
                        ↪  university.User.ID
615                     university.Offers.AboutURL =
                        ↪  input.Offers.AboutURL
```

```
616                     university.Offers.BaseURL =
                        ↪  input.Offers.BaseURL
617                     university.Offers.DeleteURL =
                        ↪  input.Offers.DeleteURL
618                     if created || input.Offers.GenerateSecret
                        ↪  {
619                             university.Offers.Secret =
                                ↪  secretOffers.Private
620                     }
621                     if created ||
                        ↪  input.Habilitations.GenerateSecret {
622                             university.Habilitations.Secret =
                                ↪  secretHabilitations.Private
623                     }
624                     localErr = db.Set(models.TableUniversity,
                        ↪  id, &university)
625             }
626             if localErr == nil && updateUser {
627                     var user models.User
628                     localErr = db.Get(models.TableUser,
                        ↪  userRef.ID, &user)
629                     if localErr == nil {
630                             index := -1
631                             for i, foreign := range
                                ↪  user.Universities {
632                                     if foreign.ID ==
                                        ↪  university.ID {
633                                             index = i
634                                             break
635                                     }
636                             }
637                             newForeign :=
                                ↪  models.UserUniversity{
638                                     ID:      university.ID,
639                                     Name:    university.Name,
640                                     Acronym:
                                        ↪  university.Acronym,
641                                     Public:
                                        ↪  university.Public,
```

```
642                                         }
643                                         if index == -1 {
644                                                 index =
                                                    ↪ len(user.Universities)
645                                                 user.Universities =
                                                    ↪ append(user.Universities,
                                                    ↪ models.UserUniversity{})
646                                         }
647                                         user.Universities[index] =
                                            ↪ newForeign
648                                         localErr =
                                            ↪ db.Set(models.TableUser,
                                            ↪ userRef.ID, &user)
649                                 }
650                         }
651                         if localErr == nil {
652                                 university.Offers.Secret =
                                    ↪ secretOffers.Public
653                                 university.Habilitations.Secret =
                                    ↪ secretHabilitations.Public
654                                 result = &university
655                         }
656                         return localErr
657                 })
658         }
659         return result, err
660 }
661
662 func (r mutationResolver) SetUser(ctx context.Context, input
    ↪ graphql.UserInput) (*models.User, error) {
663         var store kv.Store
664         var result *models.User
665         var userKey string
666         auth := r.handler.GetAuthBoss()
667         req := r.handler.GetRequest(ctx)
668         userInstance, err := auth.CurrentUser(req)
669         if err == authboss.ErrUserNotFound || err == nil && userInstance
            ↪ == nil {
670                 err = errAccessDenied
```

```
671         }
672         if err == nil && input.Password != nil {
673                 userModel := userInstance.(*models.User)
674                 if input.Password.Password !=
                    ↪  input.Password.ConfirmPassword {
675                         err = errors.New("Passwords don't match")
676                 }
677                 if authboss.VerifyPassword(userModel,
                    ↪  input.Password.OldPassword) != nil {
678                         err = errors.New("Old password don't match")
679                 }
680         }
681         if err == nil {
682                 userKey, err =
                    ↪  keygen.GenerateUserID(userInstance.GetPID())
683         }
684         if err == nil {
685                 store, err = r.handler.GetStore(ctx)
686         }
687         if err == nil {
688                 refs := []kv.Reference{
689                         {Table: models.TableUser, Key: userKey},
690                 }
691                 err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪  error {
692                         var user models.User
693                         localErr := db.Get(models.TableUser, userKey,
                            ↪  &user)
694                         if localErr == nil {
695                                 user.Name = input.Name
696                                 if input.University != nil {
697                                         user.University =
                                            ↪  *input.University
698                                 }
699                                 user.Courses = make([]models.ForeignKey,
                                    ↪  len(input.Courses))
700                                 for i, course := range input.Courses {
701                                         user.Courses[i] = *course
702                                 }
```

```
703                                 user.Habilitations =
                                    ↪ make([]models.ForeignKey,
                                    ↪ len(input.Habilitations))
704                                 for i, habilitation := range
                                    ↪ input.Habilitations {
705                                     user.Habilitations[i] =
                                        ↪ *habilitation
706                                 }
707                                 user.CompletedDisciplines =
                                    ↪ make([]models.UserDiscipline,
                                    ↪ len(input.CompletedDisciplines))
708                                 for i, discipline := range
                                    ↪ input.CompletedDisciplines {
709                                     user.CompletedDisciplines[i] =
                                        ↪ *discipline
710                                 }
711                                 if localErr == nil && input.Password !=
                                    ↪ nil &&
712                                     authboss.VerifyPassword(&user,
                                        ↪ input.Password.OldPassword)
                                        ↪ != nil {
713                                     localErr = errors.New("Old
                                        ↪ password don't match")
714                                 }
715                                 localErr = db.Set(models.TableUser,
                                    ↪ userKey, &user)
716                         }
717                         if localErr == nil {
718                                 result = &user
719                         }
720                         return localErr
721                 })
722                 if err == nil && input.Password != nil {
723                         err = auth.UpdatePassword(ctx, result,
                            ↪ input.Password.Password)
724                 }
725         }
726         return result, err
727 }
```

```
728
729 func (r mutationResolver) DeleteUser(ctx context.Context, password
    ↪  string) (bool, error) {
730         var store kv.Store
731         auth := r.handler.GetAuthBoss()
732         req := r.handler.GetRequest(ctx)
733         userInstance, err := auth.CurrentUser(req)
734         if err == authboss.ErrUserNotFound || err == nil && userInstance
            ↪  == nil {
735                 err = errAccessDenied
736         }
737         var userKey string
738         if err == nil {
739                 userKey, err =
                    ↪  keygen.GenerateUserID(userInstance.GetPID())
740         }
741         if err == nil {
742                 userModel := userInstance.(*models.User)
743                 if authboss.VerifyPassword(userModel, password) != nil {
744                         err = errAccessDenied
745                 }
746         }
747         if err == nil {
748                 store, err = r.handler.GetStore(ctx)
749         }
750         var planIDs []string
751         if err == nil {
752                 refs := []kv.Reference{
753                         {Table: models.TableUser, Key: userKey},
754                 }
755                 err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪  error {
756                         var user models.User
757                         localErr := db.Get(models.TableUser, userKey,
                            ↪  &user)
758                         if localErr == nil &&
                            ↪  authboss.VerifyPassword(&user, password) !=
                            ↪  nil {
759                                 localErr = errAccessDenied
```

```
760                         }
761                         if localErr == nil {
762                                 for _, plan := range user.Plans {
763                                         planIDs = append(planIDs,
                                        ↪  plan.ID)
764                                 }
765                                 localErr = db.Del(models.TableUser,
                                    ↪  userKey)
766                         }
767                         return localErr
768                 })
769         }
770         if err == nil {
771                 plans := make(map[string]models.Plan, len(planIDs))
772                 errorResults := store.GetMulti(models.TableUser, planIDs,
                    ↪  plans)
773                 var versionsToDelete []string
774                 for _, planID := range planIDs {
775                         if kv.IsKeyNotFoundError(errorResults[planID]) {
776                                 continue
777                         }
778                         if err == nil {
779                                 err = errorResults[planID]
780                         }
781                         if err == nil && plans[planID].User.ID != userKey
                            ↪  {
782                                 err = errAccessDenied
783                         }
784                         if err == nil {
785                                 for _, version :=
                                    ↪  range plans[planID].Versions {
786                                         versionsToDelete =
                                            ↪  append(versionsToDelete,
                                            ↪  version.ID)
787                                 }
788                                 err = store.Del(models.TablePlan, planID)
789                         }
790                 }
791                 if err == nil {
```

```
792                         err = r.deletePlanVersions(store,
                            ↪  versionsToDelete)
793                 }
794         }
795         return err == nil, err
796 }
```

Arquivo offline_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5          "sync"
6
7          "github.com/fjorgemota/gurudamatricula/graphql"
8          "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
9
10         "github.com/fjorgemota/gurudamatricula/models"
11         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
12
13         "github.com/fjorgemota/gurudamatricula/robot/importer"
14         "github.com/fjorgemota/gurudamatricula/util/indexer"
15         "github.com/fjorgemota/gurudamatricula/util/kv"
16 )
17
18 type disciplineData struct {
19         OfferIDs        []string
20         HabilitationIDs []string
21 }
22
23 var zero struct{}
24
25 func getOfflineMapStringString(versions []*graphql.OfflineIDVersion)
   ↪  map[string]string {
26         result := make(map[string]string, len(versions))
27         for _, version := range versions {
28                 result[version.ID] = version.Version
29         }
30         return result
```

```
31 }
32
33 func getOfflineToDelete(versions map[string]string) map[string]struct{} {
34         result := make(map[string]struct{}, len(versions))
35         for key := range versions {
36                 result[key] = zero
37         }
38         return result
39 }
40
41 func toOfflineSlice(result []string, toDelete map[string]struct{})
   ↪  []string {
42         for id := range toDelete {
43                 result = append(result, id)
44         }
45         return result
46 }
47
48 func getOfflineDisciplines(ctx context.Context, handler Handler, req
   ↪  graphql.OfflineRequest, resp *graphql.OfflineResponse)
   ↪  (disciplineData, error) {
49         disciplinesIndex, err := handler.GetIndex(ctx,
           ↪  importer.IndexData{
50                 ParentID: req.UniversityID,
51                 Target:   ddiffer.TargetDiscipline,
52         })
53         var disciplinesIt indexer.IndexIterator
54         if err == nil {
55                 disciplinesIt =
                   ↪  disciplinesIndex.Search(indexer.FieldTerminal{
56                         Attribute: "course_id",
57                         Value:     req.CourseID,
58                 })
59         }
60         disciplineVersions := getOfflineMapStringString(req.Disciplines)
61         toDelete := getOfflineToDelete(disciplineVersions)
62         var toLoad []string
63         offerIDs := make(map[string]struct{})
64         habilitationIDs := make(map[string]struct{})
```

```
65          for err == nil {
66                  var result models.BasicDiscipline
67                  err = disciplinesIt.Next(&result)
68                  if err == nil {
69                          delete(toDelete, result.ID)
70                          habilitationIDs[result.Habilitation.ID] = zero
71                          offerIDs[result.GenericID] = zero
72                  }
73                  if err == nil && disciplineVersions[result.ID] !=
                    ↪  result.Version {
74                          // Inserted or modified! Treat it here :D
75                          toLoad = append(toLoad, result.ID)
76                  }
77          }
78          if indexer.IsDoneError(err) {
79                  err = nil
80                  resp.DisciplinesToDelete =
                    ↪  toOfflineSlice(resp.DisciplinesToDelete, toDelete)
81                  loader := dataloaders.GetDisciplineLoader(ctx)
82                  result, errors := loader.LoadAll(toLoad)
83                  for i, key := range toLoad {
84                          if kv.IsKeyNotFoundError(errors[i]) {
85                                  // If the key was deleted..ignore it and
                                    ↪  add to habilitations to
86                                  // delete
87                                  resp.DisciplinesToDelete =
                                    ↪  append(resp.DisciplinesToDelete, key)
88                                  continue
89                          }
90                          if err == nil && errors[i] == nil &&
                            ↪  result[i].Version != disciplineVersions[key]
                            ↪  {
91                                  resp.Disciplines =
                                    ↪  append(resp.Disciplines, result[i])
92                          } else {
93                                  err = errors[i]
94                          }
95                  }
96          }
```

```
97          var result disciplineData
98          if err == nil {
99                  result.OfferIDs = toOfflineSlice(result.OfferIDs,
                    ↪  offerIDs)
100                 result.HabilitationIDs =
                    ↪  toOfflineSlice(result.HabilitationIDs,
                    ↪  habilitationIDs)
101         }
102         return result, err
103 }
104
105 func getOfflineOffers(ctx context.Context, handler Handler, req
    ↪  graphql.OfflineRequest, offerIDs []string, resp
    ↪  *graphql.OfflineResponse) error {
106         offersIndex, err := handler.GetIndex(ctx, importer.IndexData{
107                 ParentID: req.PeriodID,
108                 Target:   ddiffer.TargetDisciplineOffer,
109         })
110         offersExpressions := make([]indexer.Expression, len(offerIDs))
111         for i, genericID := range offerIDs {
112                 offersExpressions[i] = indexer.FieldTerminal{
113                         Attribute: "generic_id",
114                         Value:     genericID,
115                 }
116         }
117         disciplineOfferVersions :=
            ↪  getOfflineMapStringString(req.DisciplineOffers)
118         toDelete := getOfflineToDelete(disciplineOfferVersions)
119         var toLoad []string
120         var offersIt indexer.IndexIterator
121         if err == nil {
122                 offersIt = offersIndex.Search(indexer.Or{Operations:
                    ↪  offersExpressions})
123         }
124         for err == nil {
125                 var result models.BasicDisciplineOffer
126                 err = offersIt.Next(&result)
127                 if err == nil {
128                         delete(toDelete, result.ID)
```

```
129                 }
130                 if err == nil && disciplineOfferVersions[result.ID] !=
                    ↪  result.Version {
131                         // Inserted or modified! Treat it here :D
132                         toLoad = append(toLoad, result.ID)
133                 }
134         }
135         if indexer.IsDoneError(err) {
136                 err = nil
137                 resp.DisciplineOffersToDelete =
                    ↪  toOfflineSlice(resp.DisciplineOffersToDelete,
                    ↪  toDelete)
138                 loaders := dataloaders.GetDisciplineOfferLoader(ctx)
139                 result, errors := loaders.LoadAll(toLoad)
140                 for i, key := range toLoad {
141                         if kv.IsKeyNotFoundError(errors[i]) {
142                                 // If the key was deleted..ignore it and
                                    ↪  add to habilitations to
143                                 // delete
144                                 resp.DisciplineOffersToDelete =
                                    ↪  append(resp.DisciplineOffersToDelete,
                                    ↪  key)
145                                 continue
146                         }
147                         if err == nil && errors[i] == nil &&
                            ↪  result[i].Version !=
                            ↪  disciplineOfferVersions[key] {
148                                 resp.DisciplineOffers =
                                    ↪  append(resp.DisciplineOffers,
                                    ↪  result[i])
149                         } else {
150                                 err = errors[i]
151                         }
152                 }
153         }
154         return err
155 }
156
```

```
157 func getOfflineHabilitations(ctx context.Context, req
    ↪  graphql.OfflineRequest, habilitationIDs []string, resp
    ↪  *graphql.OfflineResponse) error {
158         var err error
159         habilitationVersions :=
            ↪  getOfflineMapStringString(req.Habilitations)
160         reqHabilitations := getOfflineToDelete(habilitationVersions)
161         for _, habilitationID := range habilitationIDs {
162                 delete(reqHabilitations, habilitationID)
163         }
164         resp.HabilitationsToDelete =
            ↪  toOfflineSlice(resp.HabilitationsToDelete, reqHabilitations)
165         loader := dataloaders.GetHabilitationLoader(ctx)
166         result, errors := loader.LoadAll(habilitationIDs)
167         for i, key := range habilitationIDs {
168                 if kv.IsKeyNotFoundError(errors[i]) {
169                         // If the key was deleted..ignore it and add to
                        ↪  habilitations to
170                         // delete
171                         resp.HabilitationsToDelete =
                        ↪  append(resp.HabilitationsToDelete, key)
172                         continue
173                 }
174                 if err == nil && errors[i] == nil {
175                         if result[i].Version == habilitationVersions[key]
                        ↪  {
176                                 // If the database still has the same
                                ↪  version as the user informed
177                                 // we can ignore it, because the user
                                ↪  already has that version saved
178                                 continue
179                         }
180                         resp.Habilitations = append(resp.Habilitations,
                        ↪  result[i])
181                 } else if err == nil {
182                         err = errors[i]
183                 }
184         }
185         return err
```

```
186 }
187
188 type offlineResponse struct {
189         mutex sync.Mutex
190         wg    sync.WaitGroup
191         data  graphql.OfflineResponse
192         err   error
193 }
194
195 func generateOfflineDump(ctx context.Context, handler Handler, req
    ↪  graphql.OfflineRequest, response *offlineResponse) {
196         defer response.wg.Done()
197         var resp graphql.OfflineResponse
198         data, err := getOfflineDisciplines(ctx, handler, req, &resp)
199         if err == nil {
200                 err = getOfflineOffers(ctx, handler, req, data.OfferIDs,
                    ↪  &resp)
201         }
202         if err == nil {
203                 err = getOfflineHabilitations(ctx, req,
                    ↪  data.HabilitationIDs, &resp)
204         }
205         response.mutex.Lock()
206         defer response.mutex.Unlock()
207         if response.err == nil {
208                 response.err = err
209         }
210         if response.err == nil {
211                 response.data.DisciplineOffers =
                    ↪  append(response.data.DisciplineOffers,
                    ↪  resp.DisciplineOffers...)
212                 response.data.DisciplineOffersToDelete =
                    ↪  append(response.data.DisciplineOffersToDelete,
                    ↪  resp.DisciplineOffersToDelete...)
213                 response.data.Disciplines =
                    ↪  append(response.data.Disciplines,
                    ↪  resp.Disciplines...)
```

```
214                 response.data.DisciplinesToDelete =
                    ↪ append(response.data.DisciplinesToDelete,
                    ↪ resp.DisciplinesToDelete...)
215                 response.data.Habilitations =
                    ↪ append(response.data.Habilitations,
                    ↪ resp.Habilitations...)
216                 response.data.HabilitationsToDelete =
                    ↪ append(response.data.HabilitationsToDelete,
                    ↪ resp.HabilitationsToDelete...)
217         }
218 }
219
220 func generateOfflineDumps(ctx context.Context, handler Handler, requests
    ↪ []*graphql.OfflineRequest) (graphql.OfflineResponse, error) {
221         var response offlineResponse
222         var err error
223         frequency := make(map[string]int)
224         for _, request := range requests {
225                 if err == nil && request != nil {
226                         response.wg.Add(1)
227                         go generateOfflineDump(ctx, handler, *request,
                            ↪ &response)
228                         disciplines :=
                            ↪ getOfflineMapStringString(request.DisciplineOffers)
229                         for discipline := range disciplines {
230                                 frequency[discipline]++
231                         }
232                 }
233         }
234         response.wg.Wait()
235         var zero struct{}
236         found := make(map[string]struct{})
237         newDisciplineOffers := response.data.DisciplineOffers[:0]
238         for _, item := range response.data.DisciplineOffers {
239                 if _, ok := found[item.ID]; !ok {
240                         // Only insert item if it's not duplicated
241                         newDisciplineOffers = append(newDisciplineOffers,
                            ↪ item)
242                         found[item.ID] = zero
```

```
243                 }
244         }
245         response.data.DisciplineOffers = newDisciplineOffers
246         for _, key := range response.data.DisciplineOffersToDelete {
247                 frequency[key]--
248         }
249         response.data.DisciplineOffersToDelete =
        ↪  response.data.DisciplineOffersToDelete[:0]
250         for key, value := range frequency {
251                 if _, ok := found[key]; !ok && value == 0 {
252                         // Only insert item if it's not in
                        ↪  DisciplineOffers list and if it should
253                         // be, in fact, deleted
254                         response.data.DisciplineOffersToDelete =
                        ↪  append(response.data.DisciplineOffersToDelete,
                        ↪  key)
255                 }
256         }
257         return response.data, response.err
258 }
```

Arquivo period_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/models"
7  )
8
9  type periodResolver struct {
10         handler Handler
11 }
12
13 func (r periodResolver) University(ctx context.Context, obj
   ↪  *models.Period) (*models.University, error) {
14         return loadUniversity(ctx, obj.UniversityID)
15 }
16
```

```
17 func (r periodResolver) Campi(ctx context.Context, obj *models.Period)
   ↪  ([]*models.Campus, error) {
18         return loadCampi(ctx, obj.Campi)
19 }
```

Arquivo plan_discipline_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/graphql"
7
8          "github.com/fjorgemota/gurudamatricula/models"
9  )
10
11 type planDisciplineResolver struct {
12         handler Handler
13 }
14
15 func (r planDisciplineResolver) University(ctx context.Context, obj
   ↪  *graphql.Discipline) (*models.University, error) {
16         return loadUniversity(ctx, obj.UniversityID)
17 }
18
19 func (r planDisciplineResolver) Course(ctx context.Context, obj
   ↪  *graphql.Discipline) (*models.Course, error) {
20         return loadCourse(ctx, obj.Course)
21 }
22
23 func (r planDisciplineResolver) Habilitation(ctx context.Context, obj
   ↪  *graphql.Discipline) (*models.Habilitation, error) {
24         return loadHabilitation(ctx, obj.Habilitation)
25 }
26
27 func (r planDisciplineResolver) Step(ctx context.Context, obj
   ↪  *graphql.Discipline) (*graphql.Step, error) {
28         return &graphql.Step{
29                 HabilitationID: obj.Habilitation.ID,
```

```
30                  ID:             obj.Step.ID,
31                  Name:           obj.Step.Name,
32          }, nil
33  }
34
35  func (r planDisciplineResolver) Tables(ctx context.Context, obj
    ↪  *graphql.Discipline) ([]*graphql.Table, error) {
36          return resolveTables(obj.Tables, obj.TableColumns, obj.TableRows)
37  }
38
39  type planDisciplineSummaryResolver struct {
40          handler Handler
41  }
42
43  func (r planDisciplineSummaryResolver) Course(ctx context.Context, obj
    ↪  *graphql.DisciplineSummary) (*models.Course, error) {
44          return loadCourse(ctx, obj.Course)
45  }
46  func (r planDisciplineSummaryResolver) Habilitation(ctx context.Context,
    ↪  obj *graphql.DisciplineSummary) (*models.Habilitation, error) {
47          return loadHabilitation(ctx, obj.Habilitation)
48  }
49  func (r planDisciplineSummaryResolver) Step(ctx context.Context, obj
    ↪  *graphql.DisciplineSummary) (*graphql.Step, error) {
50          return &graphql.Step{
51                  HabilitationID: obj.Habilitation.ID,
52                  ID:             obj.Step.ID,
53                  Name:           obj.Step.Name,
54          }, nil
55  }
```

Arquivo plan_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5          "errors"
6          "strings"
7
```

```
8         "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
9
10        "github.com/fjorgemota/gurudamatricula/graphql"
11        "github.com/fjorgemota/gurudamatricula/models"
12        "github.com/fjorgemota/gurudamatricula/util/kv"
13)
14
15type planVersionResolver struct {
16        handler Handler
17}
18
19func (r planVersionResolver) Disciplines(ctx context.Context, obj
  ↪  *models.PlanVersion) ([]*graphql.PlanDiscipline, error) {
20        var result []*graphql.PlanDiscipline
21        var err error
22        var ids []string
23        indexByIds := make(map[string]int)
24        for _, discipline := range obj.Disciplines {
25                if _, ok := indexByIds[discipline.ID]; !ok {
26                        indexByIds[discipline.ID] = len(ids)
27                        ids = append(ids, discipline.ID)
28                }
29        }
30        loader := dataloaders.GetDisciplineLoader(ctx)
31        disciplines, errors := loader.LoadAll(ids)
32        for disciplineID, discipline := range obj.Disciplines {
33                planDiscipline := graphql.PlanDiscipline{
34                        Discipline: &graphql.Discipline{
35                                ID:           discipline.ID,
36                                GenericID:    discipline.GenericID,
37                                UniversityID: obj.University.ID,
38                                Version:      discipline.Version,
39                                Code:         discipline.Code,
40                                Name:         discipline.Name,
41                                Course:       discipline.Course,
42                                Habilitation: discipline.Habilitation,
43                                Step:         discipline.Step,
44                                Description:  discipline.Description,
45                                Type:         discipline.Type,
```

```
46                      },
47              }
48              position := indexByIds[discipline.ID]
49              if errors[position] == nil &&
                ↪  disciplines[position].Version != discipline.Version {
50                      planDiscipline.Updated = disciplines[position]
51              } else if kv.IsKeyNotFoundError(errors[position]) {
52                      planDiscipline.Deleted = true
53              } else if err == nil && errors[position] != nil {
54                      err = errors[position]
55              }
56              indexes := make(map[int]int)
57
58              for _, related := range obj.DisciplinesRelatedDisciplines
                ↪  {
59                      if related.DisciplineID == disciplineID {
60                              planDiscipline.Discipline.RelatedDisciplines
                                ↪  =
                                ↪  append(planDiscipline.Discipline.RelatedDisciplines,
                                ↪  graphql.DisciplineSummary{
61                                      ID:           related.ID,
62                                      GenericID:    related.GenericID,
63                                      UniversityID: obj.University.ID,
64                                      Version:      related.Version,
65                                      Code:         related.Code,
66                                      Name:         related.Name,
67                                      Course:       related.Course,
68                                      Habilitation:
                                        ↪  related.Habilitation,
69                                      Step:         related.Step,
70                                      Description:
                                        ↪  related.Description,
71                                      Type:         related.Type,
72                              })
73                      }
74              }
75              for relatedID, related := range obj.DisciplinesRelated {
76                      if related.DisciplineID == disciplineID {
```

```
77                                  indexes[relatedID] =
                                    ↪ len(planDiscipline.Discipline.Related)
78                                  planDiscipline.Discipline.Related =
                                    ↪ append(planDiscipline.Discipline.Related,
                                    ↪ graphql.DisciplineRelated{
79                                      Name: related.Name,
80                                      Type:
                                        ↪ graphql.DisciplineRelatedType(strings.T
81                                  })
82                          }
83                  }
84                  for _, item := range obj.DisciplinesRelatedItem {
85                          if relatedID, ok := indexes[item.RelatedID]; ok {
86                                  var description *string
87                                  if len(item.Description) > 0 {
88                                          description = &item.Description
89                                  }
90                                  planDiscipline.Discipline.Related[relatedID].Items
                                    ↪ =
                                    ↪ append(planDiscipline.Discipline.Related[relate
                                    ↪ &graphql.DisciplineRelatedItem{
91                                      Type:
                                        ↪ graphql.DisciplineRelatedItemType(strin
92                                      Description: description,
93                                      Value:       item.Value,
94                                  })
95                          }
96                  }
97                  for tableID, teamTable := range obj.Tables {
98                          if teamTable.Type == models.DisciplineTableType
                            ↪ && teamTable.Index == disciplineID {
99                                  newTableID :=
                                    ↪ len(planDiscipline.Discipline.Tables)
100                                 planDiscipline.Discipline.Tables =
                                    ↪ append(planDiscipline.Discipline.Tables,
                                    ↪ models.Table{
101                                     ID:          teamTable.ID,
102                                     Title:       teamTable.Title,
```

```
103                             Description:
                                ↪  teamTable.Description,
104                         })
105                         for index := range indexes {
106                             delete(indexes, index)
107                         }
108                         for columnID, column := range
                            ↪  obj.TableColumns {
109                             if column.TableID == tableID {
110                                 indexes[columnID] =
                                    ↪  len(planDiscipline.Discipline.TableC
111                                 planDiscipline.Discipline.TableColumns
                                    ↪  =
                                    ↪  append(planDiscipline.Discipline.Tab
                                    ↪  models.TableColumn{
112                                     ID:
                                        ↪  column.ID,
113                                     Name:
                                        ↪  column.Name,
114                                     TableID:
                                        ↪  newTableID,
115                                 })
116                             }
117                         }
118                         for _, row := range obj.TableRows {
119                             if column, ok :=
                                ↪  indexes[row.Column]; ok {
120                                 planDiscipline.Discipline.TableRows
                                    ↪  =
                                    ↪  append(planDiscipline.Discipline.Tab
                                    ↪  models.TableRow{
121                                     Row:    row.Row,
122                                     Column: column,
123                                     Value:
                                        ↪  row.Value,
124                                 })
125                             }
126                         }
127                 }
```

```
128                 }
129                 result = append(result, &planDiscipline)
130         }
131         return result, err
132 }
133
134 func (r planVersionResolver) SelectedSchedules(ctx context.Context, obj
    ↪  *models.PlanVersion) ([]*graphql.PlanSelectedSchedule, error) {
135         var result []*graphql.PlanSelectedSchedule
136         for _, possibilityTeam := range obj.PossibilitiesTeams {
137                 if possibilityTeam.PossibilityID ==
                    ↪  obj.SelectedPossibility && possibilityTeam.Enabled &&
                    ↪  possibilityTeam.Selected {
138                         for _, teamSchedule := range obj.TeamSchedules {
139                                 if teamSchedule.TeamID ==
                                    ↪  possibilityTeam.TeamID {
140                                         schedule :=
                                            ↪  obj.Schedules[teamSchedule.ScheduleID]
141                                         team :=
                                            ↪  obj.Teams[teamSchedule.TeamID]
142                                         disciplineOffer :=
                                            ↪  obj.DisciplinesOffers[team.DisciplineOf
143                                         result = append(result,
                                            ↪  &graphql.PlanSelectedSchedule{
144                                                 Start:
                                                    ↪  &schedule.Start,
145                                                 End:       &schedule.End,
146                                                 DayOfWeek:
                                                    ↪  int(schedule.DayOfWeek),
147                                                 OfferID:
                                                    ↪  disciplineOffer.ID,
148                                                 Title:
                                                    ↪  disciplineOffer.Code,
149                                                 Color:
                                                    ↪  possibilityTeam.Color,
150                                         })
151                                 }
152                         }
153                 }
```

```
154         }
155         return result, nil
156 }
157
158 func (r planVersionResolver) DisciplineOffers(ctx context.Context, obj
    ↪  *models.PlanVersion) ([]*graphql.PlanDisciplineOffer, error) {
159         var result []*graphql.PlanDisciplineOffer
160         var ids []string
161         var err error
162         indexByIds := make(map[string]int)
163         for _, discipline := range obj.DisciplinesOffers {
164                 if _, ok := indexByIds[discipline.ID]; !ok &&
                    ↪  !discipline.Custom {
165                         indexByIds[discipline.ID] = len(ids)
166                         ids = append(ids, discipline.ID)
167                 }
168         }
169         loader := dataloaders.GetDisciplineOfferLoader(ctx)
170         offers, errors := loader.LoadAll(ids)
171         for offerID, discipline := range obj.DisciplinesOffers {
172                 planDisciplineOffer := graphql.PlanDisciplineOffer{
173                         Custom: discipline.Custom,
174                         Offer: &models.DisciplineOffer{
175                                 ID:           discipline.ID,
176                                 UniversityID: obj.University.ID,
177                                 Period:       obj.Period,
178                                 Campus:       discipline.Campus,
179                                 Version:      discipline.Version,
180                                 Code:         discipline.Code,
181                                 Name:         discipline.Name,
182                         },
183                 }
184                 if !discipline.Custom {
185                         position := indexByIds[discipline.ID]
186                         if errors[position] == nil &&
                            ↪  offers[position].Version !=
                            ↪  discipline.Version {
187                                 planDisciplineOffer.Updated =
                                    ↪  offers[position]
```

```
188                         } else if kv.IsKeyNotFoundError(errors[position])
                            ↪ {
189                                 planDisciplineOffer.Deleted = true
190                         } else if err == nil && errors[position] != nil {
191                                 err = errors[position]
192                         }
193                 }
194                 schedulesIndex :=
                    ↪ make(map[models.DisciplineOfferSchedule]int)
195                 roomsIndex := make(map[models.DisciplineOfferRoom]int)
196                 teachersIndex :=
                    ↪ make(map[models.DisciplineOfferTeacher]int)
197                 for teamID, team := range obj.Teams {
198                         if err == nil && team.DisciplineOfferID ==
                            ↪ offerID {
199                                 newTeamID :=
                                    ↪ len(planDisciplineOffer.Offer.Teams)
200                                 planDisciplineOffer.Offer.Teams =
                                    ↪ append(planDisciplineOffer.Offer.Teams,
                                    ↪ models.DisciplineOfferTeam{
201                                     ID:        team.ID,
202                                     Code:      team.Code,
203                                     Course:    team.Course,
204                                     Vacancies: team.Vacancies,
205                                 })
206                                 for _, teamSchedule := range
                                    ↪ obj.TeamSchedules {
207                                     if teamSchedule.TeamID == teamID
                                        ↪ {
208                                         schedule :=
                                            ↪ obj.Schedules[teamSchedule.Sche
209                                         if _, ok :=
                                            ↪ schedulesIndex[schedule];
                                            ↪ !ok {
210                                             schedulesIndex[schedule]
                                                ↪ =
                                                ↪ len(planDisciplineOffer
```

```
211                                                 planDisciplineOffer.Offer.Schedu
                                                    ↪ =
                                                    ↪ append(planDisciplineOffer.O
                                                    ↪ schedule)
212                                             }
213                                             newTeamSchedule :=
                                                ↪ models.DisciplineOfferTeamSchedule{
214                                                 ScheduleID:
                                                    ↪ schedulesIndex[schedule],
215                                                 TeamID:
                                                    ↪ newTeamID,
216                                                 RoomID:     -1,
217                                             }
218                                             if teamSchedule.RoomID !=
                                                ↪ -1 {
219                                                 room :=
                                                    ↪ obj.Rooms[teamSchedule.RoomI
220                                                 if _, ok :=
                                                    ↪ roomsIndex[room];
                                                    ↪ !ok {
221                                                     roomsIndex[room]
                                                        ↪ =
                                                        ↪ len(planDisciplineOf
222                                                     planDisciplineOffer.Offe
                                                        ↪ =
                                                        ↪ append(planDisciplin
                                                        ↪ room)
223                                                 }
224                                                 newTeamSchedule.RoomID
                                                    ↪ =
                                                    ↪ roomsIndex[room]
225                                             }
226                                             planDisciplineOffer.Offer.TeamSchedules
                                                ↪ =
                                                ↪ append(planDisciplineOffer.Offer.Tea
                                                ↪ newTeamSchedule)
227                                     }
228                             }
```

```
229                             for _, teamTeacher := range
                                ↪  obj.TeamTeachers {
230                                 if teamTeacher.TeamID == teamID {
231                                         teacher :=
                                            ↪  obj.Teachers[teamTeacher.Teache
232                                         if _, ok :=
                                            ↪  teachersIndex[teacher];
                                            ↪  !ok {
233                                             teachersIndex[teacher]
                                                ↪  =
                                                ↪  len(planDisciplineOffer
234                                             planDisciplineOffer.Offer.T
                                                ↪  =
                                                ↪  append(planDisciplineOf
                                                ↪  teacher)
235                                         }
236                                         planDisciplineOffer.Offer.TeamTeach
                                            ↪  =
                                            ↪  append(planDisciplineOffer.Offe
                                            ↪  models.DisciplineOfferTeamTeach
237                                             TeacherID:
                                                ↪  teachersIndex[teacher],
238                                             TeamID:
                                                ↪  newTeamID,
239                                         })
240                                 }
241                             }
242                             for tableID, teamTable := range
                                ↪  obj.Tables {
243                                 if teamTable.Type ==
                                    ↪  models.TeamTableType &&
                                    ↪  teamTable.Index == teamID {
244                                         newTableID :=
                                            ↪  len(planDisciplineOffer.Offer.T
245                                         planDisciplineOffer.Offer.TeamTable
                                            ↪  =
                                            ↪  append(planDisciplineOffer.Offe
                                            ↪  models.DisciplineOfferTeamTable
```

```
246                         TeamID:
                            ↪ newTeamID,
247                         ID:
                            ↪ teamTable.ID,
248                         Title:
                            ↪ teamTable.Title,
249                         Description:
                            ↪ teamTable.Description,
250                 })
251                 columnsIndex :=
                    ↪ make(map[int]int)
252                 for columnID, column :=
                    ↪ range
                    ↪ obj.TableColumns {
253                         if column.TableID
                            ↪ == tableID {
254                                 columnsIndex[columnID]
                                    ↪ =
                                    ↪ len(planDisciplineOf
255                                 planDisciplineOffer.Offe
                                    ↪ =
                                    ↪ append(planDisciplin
                                    ↪ models.TableColumn{
256                                     ID:
                                        ↪ column.ID,
257                                     Name:
                                        ↪ column.Name,
258                                     TableID:
                                        ↪ newTableID,
259                                 })
260                         }
261                 }
262                 for _, row := range
                    ↪ obj.TableRows {
263                         if column, ok :=
                            ↪ columnsIndex[row.Column];
                            ↪ ok {
```

```
264                                                                 planDisciplineOffer
                                                                    ↪ =
                                                                    ↪ append(planDisc
                                                                    ↪ models.TableRow
265                                                                     Row:
                                                                        ↪ row.Row
266                                                                     Column:
                                                                        ↪ column,
267                                                                     Value:
                                                                        ↪ row.Val
268                                                                 })
269                                                             }
270                                                         }
271                                                     }
272                                                 }
273                                             }
274                                     }
275                                     result = append(result, &planDisciplineOffer)
276                             }
277                             return result, err
278     }
279
280     func (r planVersionResolver) Possibilities(ctx context.Context, obj
        ↪ *models.PlanVersion) ([]*graphql.PlanPossibility, error) {
281             var result []*graphql.PlanPossibility
282             for possibilityID, possibility := range obj.Possibilities {
283                     planPossibility := graphql.PlanPossibility{
284                             Name: possibility.Name,
285                     }
286                     for _, possibilityTeam := range obj.PossibilitiesTeams {
287                             if possibilityTeam.PossibilityID == possibilityID
                                ↪ {
288                                     team := obj.Teams[possibilityTeam.TeamID]
289                                     offer :=
                                        ↪ obj.DisciplinesOffers[team.DisciplineOfferID]
290                                     planPossibility.Teams =
                                        ↪ append(planPossibility.Teams,
                                        ↪ &graphql.PlanTeamSelection{
291                                             TeamID:       team.ID,
```

```
292                                         OfferID:      offer.ID,
293                                         OfferVersion: offer.Version,
294                                         Color:
                                            ↪  possibilityTeam.Color,
295                                         Selected:
                                            ↪  possibilityTeam.Selected,
296                                         Enabled:
                                            ↪  possibilityTeam.Enabled,
297                                 })
298                         }
299                 }
300                 for _, possibilityDiscipline := range
                    ↪  obj.PossibilitiesDisciplines {
301                         if possibilityDiscipline.PossibilityID ==
                            ↪  possibilityID {
302                                 discipline :=
                                    ↪  obj.Disciplines[possibilityDiscipline.DisciplineID]
303                                 planPossibility.Disciplines =
                                    ↪  append(planPossibility.Disciplines,
                                    ↪  &graphql.PlanDisciplineSelection{
304                                         DisciplineID:      discipline.ID,
305                                         DisciplineVersion:
                                            ↪  discipline.Version,
306                                         Enabled:
                                            ↪  possibilityDiscipline.Enabled,
307                                 })
308                         }
309                 }
310                 result = append(result, &planPossibility)
311         }
312         return result, nil
313 }
314
315 type planResolver struct {
316         handler Handler
317 }
318
319 func (r planResolver) User(ctx context.Context, obj *models.Plan)
    ↪  (*models.User, error) {
```

```
320         var result *models.User
321         var err error
322         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
323                 result = &models.User{
324                         ID:   obj.User.ID,
325                         Name: obj.User.Name,
326                 }
327         } else {
328                 loader := dataloaders.GetUserLoader(ctx)
329                 result, err = loader.Load(obj.User.ID)
330         }
331         return result, err
332 }
333
334 func (r planResolver) University(ctx context.Context, obj *models.Plan)
    ↪  (*models.University, error) {
335         var result *models.University
336         var err error
337         if answerWithSummaryOnly(ctx, []string{"id", "name"}) {
338                 result = &models.University{
339                         ID:   obj.University.ID,
340                         Name: obj.University.Name,
341                 }
342         } else {
343                 loader := dataloaders.GetUniversityLoader(ctx)
344                 result, err = loader.Load(obj.University.ID)
345         }
346         return result, err
347 }
348
349 func (r planResolver) Period(ctx context.Context, obj *models.Plan)
    ↪  (*models.Period, error) {
350         return loadPeriod(ctx, obj.Period)
351 }
352
353 func (r planResolver) Versions(ctx context.Context, obj *models.Plan,
    ↪  startArg, endArg *int) ([]*models.PlanVersion, error) {
354         var err error
```

```
355         start, end := getStartEndIndex(len(obj.Versions), startArg,
            ↪  endArg)
356         versions := obj.Versions[start:end]
357         result := make([]*models.PlanVersion, 0, len(versions))
358         onlySummary := answerWithSummaryOnly(ctx, []string{"id",
            ↪  "savedAt", "totalPossibilities", "selectedSchedules"})
359         if onlySummary {
360                 for i, version := range versions {
361                         newVersion := &models.PlanVersion{
362                                 ID:                  version.ID,
363                                 SavedAt:             version.SavedAt,
364                                 TotalPossibilities:
                                    ↪  version.TotalPossibilities,
365                                 SelectedPossibility: 0,
366                         }
367                         schedulesIndexes := make(map[int]int)
368                         versionIndex := i + start
369                         for _, versionSchedule := range
                            ↪  obj.VersionSchedules {
370                                 if versionSchedule.Version ==
                                    ↪  versionIndex &&
                                    ↪  versionSchedule.Schedule <
                                    ↪  len(obj.Schedules) {
371                                         var scheduleIndex int
372                                         var ok bool
373                                         schedule :=
                                            ↪  obj.Schedules[versionSchedule.Schedule]
374                                         if scheduleIndex, ok =
                                            ↪  schedulesIndexes[versionSchedule.Schedule];
                                            ↪  !ok {
375                                                 scheduleIndex =
                                                    ↪  len(newVersion.Schedules)
376                                                 schedulesIndexes[versionSchedule.Schedul
                                                    ↪  = scheduleIndex
377                                                 newVersion.Schedules =
                                                    ↪  append(newVersion.Schedules,
                                                    ↪  schedule.Interval)
378                                         }
```

```
379                                     offerID :=
                                        ↪ len(newVersion.DisciplinesOffers)
380                                     teamID := len(newVersion.Teams)
381                                     newVersion.Teams =
                                        ↪ append(newVersion.Teams,
                                        ↪ models.PlanVersionTeam{
382                                             DisciplineOfferID:
                                                ↪ offerID,
383                                     })
384                                     newVersion.DisciplinesOffers =
                                        ↪ append(newVersion.DisciplinesOffers,
                                        ↪ models.PlanVersionDisciplineOffer{
385                                             ID:   schedule.ID,
386                                             Code: schedule.Title,
387                                     })
388                                     newVersion.PossibilitiesTeams =
                                        ↪ append(newVersion.PossibilitiesTeams,
                                        ↪ models.PlanPossibilityTeam{
389                                             PossibilityID: 0,
390                                             OfferID:       offerID,
391                                             TeamID:        teamID,
392                                             Color:
                                                ↪ schedule.Color,
393                                             Enabled:       true,
394                                             Selected:      true,
395                                     })
396                                     newVersion.TeamSchedules =
                                        ↪ append(newVersion.TeamSchedules,
                                        ↪ models.DisciplineOfferTeamSchedule{
397                                             RoomID:     -1,
398                                             ScheduleID:
                                                ↪ scheduleIndex,
399                                             TeamID:     teamID,
400                                     })
401                             }
402                     }
403                     result = append(result, newVersion)
404             }
405     } else {
```

```
406                 ids := make([]string, len(versions))
407                 for i, version := range versions {
408                         ids[i] = version.ID
409                 }
410                 loader := dataloaders.GetPlanVersionLoader(ctx)
411                 results, errors := loader.LoadAll(ids)
412                 for i := range ids {
413                         if kv.IsKeyNotFoundError(errors[i]) {
414                                 continue
415                         }
416                         if err == nil {
417                                 err = errors[i]
418                         }
419                         if err == nil {
420                                 result = append(result, results[i])
421                         }
422                 }
423         }
424         return result, err
425 }
426
427 func (r planResolver) VersionByIndex(ctx context.Context, obj
    ↪  *models.Plan, index int) (*models.PlanVersion, error) {
428         var result *models.PlanVersion
429         next := index + 1
430         var end *int
431         if next != 0 {
432                 end = &next
433         }
434         data, err := r.Versions(ctx, obj, &index, end)
435         if len(data) == 1 && err == nil {
436                 result = data[0]
437         } else if len(data) == 0 && err == nil {
438                 err = errors.New("Invalid index")
439         }
440         return result, err
441 }
442
```

```
443 func (r planResolver) VersionByID(ctx context.Context, obj *models.Plan,
    ↪  id string) (*models.PlanVersion, error) {
444         for i, version := range obj.Versions {
445                 if version.ID == id {
446                         return r.VersionByIndex(ctx, obj, i)
447                 }
448         }
449         return nil, errors.New("Version ID not found")
450 }
```

Arquivo query_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5          "errors"
6
7          "github.com/fjorgemota/gurudamatricula/models/keygen"
8          "github.com/volatiletech/authboss"
9
10         "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
11
12         "github.com/fjorgemota/gurudamatricula/graphql"
13         "github.com/fjorgemota/gurudamatricula/models"
14         "github.com/fjorgemota/gurudamatricula/util/kv"
15 )
16
17 type queryResolver struct {
18         handler Handler
19 }
20
21 func (r queryResolver) checkIfCanAccessUniversity(ctx context.Context,
    ↪  row *models.University) (bool, error) {
22         var err error
23         authorized := row.Public
24         if !authorized && err == nil {
25                 // Only check authentication if the university is not
                   ↪  public
26                 var userID string
```

```
27                  userID, err = r.getUserID(ctx)
28                  authorized = len(userID) > 0 && row.User.ID == userID
29          }
30          return err == nil && authorized, err
31  }
32
33  func (r queryResolver) checkIfUniversityIsPublic(ctx context.Context,
    ↪  universityID string) (bool, error) {
34          loader := dataloaders.GetUniversityLoader(ctx)
35          row, err := loader.Load(universityID)
36          var authorized bool
37          if err == nil {
38                  authorized, err = r.checkIfCanAccessUniversity(ctx, row)
39          }
40          return authorized, err
41  }
42
43  func (r queryResolver) checkIfUniversitiesArePublic(ctx context.Context,
    ↪  universitiesIDs []string) (bool, error) {
44          var err error
45          universityLoader := dataloaders.GetUniversityLoader(ctx)
46          universities, errors := universityLoader.LoadAll(universitiesIDs)
47          isPublic := true
48          for i := range universitiesIDs {
49                  if err == nil {
50                          err = errors[i]
51                  }
52                  if err == nil {
53                          isPublic, err = r.checkIfCanAccessUniversity(ctx,
                            ↪  universities[i])
54                  }
55          }
56          return isPublic, err
57  }
58
59  func (r queryResolver) University(ctx context.Context, id string)
    ↪  (*models.University, error) {
60          loader := dataloaders.GetUniversityLoader(ctx)
61          row, err := loader.Load(id)
```

```
62          var authorized bool
63          if err == nil {
64                  authorized, err = r.checkIfCanAccessUniversity(ctx, row)
65          }
66          if err == nil && !authorized {
67                  err = errors.New("University not found")
68          }
69          return row, err
70  }
71
72  func (r queryResolver) Universities(ctx context.Context, onlyOwnedPtr
    ↪  *bool) ([]*models.University, error) {
73          var result []*models.University
74          store, err := r.handler.GetStore(ctx)
75          userID, err := r.getUserID(ctx)
76          onlyOwned := false
77          if onlyOwnedPtr != nil {
78                  onlyOwned = *onlyOwnedPtr
79          }
80          if err == nil {
81                  it := store.GetAll(models.TableUniversity)
82                  for err == nil {
83                          var item models.University
84                          var authorized bool
85                          _, err = it.Next(&item)
86                          if !onlyOwned || (onlyOwned && len(userID) > 0 &&
                            ↪  item.User.ID == userID) {
87                                  if err == nil {
88                                          authorized, err =
                                            ↪  r.checkIfCanAccessUniversity(ctx,
                                            ↪  &item)
89                                  }
90                                  if err == nil && authorized {
91                                          result = append(result, &item)
92                                  }
93                          }
94                  }
95                  if kv.IsDoneError(err) {
96                          err = nil
```

```
97                  }
98          }
99          return result, err
100 }
101 func (r queryResolver) Course(ctx context.Context, id string)
    ↪  (*models.Course, error) {
102         loader := dataloaders.GetCourseLoader(ctx)
103         result, err := loader.Load(id)
104         var isPublic bool
105         if err == nil {
106                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  result.UniversityID)
107         }
108         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
109                 err = errors.New("Course not found")
110         }
111         return result, err
112 }
113
114 func (r queryResolver) CoursesByUniversity(ctx context.Context,
    ↪  universityID string) ([]*models.Course, error) {
115         var courses []*models.Course
116         var authorized bool
117         universityLoader := dataloaders.GetUniversityLoader(ctx)
118         row, err := universityLoader.Load(universityID)
119         if err == nil {
120                 authorized, err = r.checkIfCanAccessUniversity(ctx, row)
121                 if err == nil && !authorized {
122                         err = errors.New("courses not found")
123                 }
124         }
125         if err == nil {
126                 courses, err = loadCourses(ctx, row.Courses)
127         }
128         return courses, err
129 }
130
131 func (r queryResolver) CoursesByIDs(ctx context.Context, ids []string)
    ↪  ([]*models.Course, error) {
```

```
132         var result []*models.Course
133         var err error
134         loader := dataloaders.GetCourseLoader(ctx)
135         result, errs := loader.LoadAll(ids)
136         universities := make([]string, len(ids))
137         for i := range ids {
138                 if kv.IsKeyNotFoundError(errs[i]) {
139                         continue
140                 }
141                 if err == nil {
142                         err = errs[i]
143                 }
144                 if err == nil {
145                         universities[i] = result[i].UniversityID
146                 }
147         }
148         var isPublic bool
149         if err == nil {
150                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                ↪  universities)
151         }
152         if err == nil && !isPublic {
153                 err = errors.New("Courses not found")
154         }
155         return result, err
156 }
157
158 func (r queryResolver) Period(ctx context.Context, id string)
    ↪  (*models.Period, error) {
159         loader := dataloaders.GetPeriodLoader(ctx)
160         result, err := loader.Load(id)
161         var isPublic bool
162         if err == nil {
163                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                ↪  result.UniversityID)
164         }
165         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
166                 err = errors.New("Period not found")
167         }
```

```
168         return result, err
169 }
170
171 func (r queryResolver) PeriodsByUniversity(ctx context.Context,
    ↪  universityID string) ([]*models.Period, error) {
172         var periods []*models.Period
173         var authorized bool
174         universityLoader := dataloaders.GetUniversityLoader(ctx)
175         row, err := universityLoader.Load(universityID)
176         if err == nil {
177                 authorized, err = r.checkIfCanAccessUniversity(ctx, row)
178                 if err == nil && !authorized {
179                         err = errors.New("Periods not found")
180                 }
181         }
182         if err == nil {
183                 periods, err = loadPeriods(ctx, row.Periods)
184         }
185         return periods, err
186 }
187
188 func (r queryResolver) PeriodsByIDs(ctx context.Context, ids []string)
    ↪  ([]*models.Period, error) {
189         var result []*models.Period
190         var err error
191         loader := dataloaders.GetPeriodLoader(ctx)
192         result, errs := loader.LoadAll(ids)
193         universities := make([]string, len(ids))
194         for i := range ids {
195                 if kv.IsKeyNotFoundError(errs[i]) {
196                         continue
197                 }
198                 if err == nil {
199                         err = errs[i]
200                 }
201                 if err == nil {
202                         universities[i] = result[i].UniversityID
203                 }
204         }
```

```
205         var isPublic bool
206         if err == nil {
207                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                    ↪ universities)
208         }
209         if err == nil && !isPublic {
210                 err = errors.New("Periods not found")
211         }
212         return result, err
213 }
214
215 func (r queryResolver) GetOfflineUpdates(ctx context.Context, requests
    ↪ []*graphql.OfflineRequest) (*graphql.OfflineResponse, error) {
216         var result *graphql.OfflineResponse
217         var response graphql.OfflineResponse
218         var err error
219         newRequests := requests[:0]
220         var deletedRequests []*graphql.OfflineRequest
221         var zero struct{}
222         universitiesIDsMap := make(map[string]struct{})
223         for _, request := range requests {
224                 universitiesIDsMap[request.UniversityID] = zero
225         }
226         universitiesIDsList := make([]string, 0, len(universitiesIDsMap))
227         for university := range universitiesIDsMap {
228                 universitiesIDsList = append(universitiesIDsList,
                    ↪ university)
229         }
230         universityLoader := dataloaders.GetUniversityLoader(ctx)
231         var universitiesList []*models.University
232         var errors []error
233         universitiesMap := make(map[string]*models.University)
234         universitiesList, errors =
            ↪ universityLoader.LoadAll(universitiesIDsList)
235         for i := range universitiesIDsList {
236                 if kv.IsKeyNotFoundError(errors[i]) {
237                         continue
238                 }
239                 if err == nil {
```

```
240                     err = errors[i]
241             }
242             if err == nil && universitiesList[i] != nil {
243                     id := universitiesList[i].ID
244                     universitiesMap[id] = universitiesList[i]
245             }
246     }
247     for _, request := range requests {
248             if err == nil {
249                     var authorized bool
250                     university :=
                        ↪  universitiesMap[request.UniversityID]
251                     if university != nil {
252                             authorized, err =
                                ↪  r.checkIfCanAccessUniversity(ctx,
                                ↪  university)
253                     }
254                     if err == nil {
255                             if authorized {
256                                     newRequests = append(newRequests,
                                        ↪  request)
257                             } else {
258                                     deletedRequests =
                                        ↪  append(deletedRequests,
                                        ↪  request)
259                             }
260                     }
261             }
262     }
263     if err == nil {
264             response, err = generateOfflineDumps(ctx, r.handler,
                ↪  newRequests)
265     }
266     if err == nil {
267             found := make(map[string]struct{},
                ↪  len(response.DisciplineOffers))
268             for _, discipline := range response.DisciplineOffers {
269                     found[discipline.ID] = zero
270             }
```

```
271                 for _, request := range deletedRequests {
272                         for _, disciplineOffer := range
                            ↪  request.DisciplineOffers {
273                                 if _, ok := found[disciplineOffer.ID];
                                    ↪  !ok {
274                                         response.DisciplineOffersToDelete
                                            ↪  =
                                            ↪  append(response.DisciplineOffersToDelet
                                            ↪  disciplineOffer.ID)
275                                 }
276                         }
277                         for _, discipline := range request.Disciplines {
278                                 response.DisciplinesToDelete =
                                    ↪  append(response.DisciplinesToDelete,
                                    ↪  discipline.ID)
279                         }
280                         for _, habilitation := range
                            ↪  request.Habilitations {
281                                 response.HabilitationsToDelete =
                                    ↪  append(response.HabilitationsToDelete,
                                    ↪  habilitation.ID)
282                         }
283                 }
284
285                 result = &response
286         }
287         return result, err
288 }
289
290 func (r queryResolver) getUserID(ctx context.Context) (string, error) {
291         req := r.handler.GetRequest(ctx)
292         auth := r.handler.GetAuthBoss()
293         result, err := auth.CurrentUserID(req)
294         if err == nil && result != "" {
295                 result, err = keygen.GenerateUserID(result)
296         }
297         if err == authboss.ErrUserNotFound {
298                 err = nil
299         }
```

```
300         return result, err
301 }
302
303 func (r queryResolver) LoggedUser(ctx context.Context) (*models.User,
    ↪  error) {
304         var result *models.User
305         req := r.handler.GetRequest(ctx)
306         auth := r.handler.GetAuthBoss()
307         user, err := auth.CurrentUser(req)
308         if err == nil && user != nil {
309                 result = user.(*models.User)
310         }
311         if err == authboss.ErrUserNotFound {
312                 err = nil
313         }
314         return result, err
315 }
316
317 func (r queryResolver) CampiByPeriod(ctx context.Context, periodID
    ↪  string) ([]*models.Campus, error) {
318         var result []*models.Campus
319         loader := dataloaders.GetPeriodLoader(ctx)
320         item, err := loader.Load(periodID)
321         var isPublic bool
322         if err == nil && item != nil {
323                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  item.UniversityID)
324         }
325         if err == nil && !isPublic {
326                 err = errors.New("Campi not found")
327         }
328         if err == nil && item != nil {
329                 result, err = loadCampi(ctx, item.Campi)
330         }
331         return result, err
332 }
333
334 func (r queryResolver) CampiByIDs(ctx context.Context, ids []string)
    ↪  ([]*models.Campus, error) {
```

```
335         var result []*models.Campus
336         var err error
337         loader := dataloaders.GetCampusLoader(ctx)
338         result, errs := loader.LoadAll(ids)
339         universities := make([]string, len(ids))
340         for i := range ids {
341                 if kv.IsKeyNotFoundError(errs[i]) {
342                         continue
343                 }
344                 if err == nil {
345                         err = errs[i]
346                 }
347                 if err == nil {
348                         universities[i] = result[i].UniversityID
349                 }
350         }
351         var isPublic bool
352         if err == nil {
353                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                ↪  universities)
354         }
355         if err == nil && !isPublic {
356                 err = errors.New("Campus not found")
357         }
358         return result, err
359 }
360
361 func (r queryResolver) Campus(ctx context.Context, id string)
    ↪  (*models.Campus, error) {
362         loader := dataloaders.GetCampusLoader(ctx)
363         var isPublic bool
364         result, err := loader.Load(id)
365         if err == nil {
366                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                ↪  result.UniversityID)
367         }
368         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
369                 err = errors.New("Campus not found")
370         }
```

```
371         return result, err
372 }
373
374 func (r queryResolver) HabilitationsByCourse(ctx context.Context,
    ↪  courseID string) ([]*models.Habilitation, error) {
375         var result []*models.Habilitation
376         loader := dataloaders.GetCourseLoader(ctx)
377         item, err := loader.Load(courseID)
378         onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name"})
379         var isPublic bool
380         if err == nil && item != nil {
381                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  item.UniversityID)
382         }
383         if err == nil && !isPublic {
384                 err = errors.New("Habilitations not found")
385         }
386         if err == nil && item != nil {
387                 result = make([]*models.Habilitation, 0,
                    ↪  len(item.Habilitations))
388                 if onlySummary {
389                         for _, Habilitation := range item.Habilitations {
390                                 result = append(result,
                                    ↪  &models.Habilitation{
391                                         ID:   Habilitation.ID,
392                                         Name: Habilitation.Name,
393                                 })
394                         }
395                 } else {
396                         keys := make([]string, len(item.Habilitations))
397                         for i, Habilitation := range item.Habilitations {
398                                 keys[i] = Habilitation.ID
399                         }
400                         loader := dataloaders.GetHabilitationLoader(ctx)
401                         results, errors := loader.LoadAll(keys)
402                         for i := range keys {
403                                 if kv.IsKeyNotFoundError(errors[i]) {
404                                         continue
405                                 }
```

```
406                                 if err == nil {
407                                         err = errors[i]
408                                 }
409                                 if err == nil && results[i] != nil {
410                                         result = append(result,
                                            ↪  results[i])
411                                 }
412                         }
413                 }
414         }
415         return result, err
416 }
417
418 func (r queryResolver) HabilitationsByCourses(ctx context.Context,
    ↪  coursesIDs []string) ([]*models.Habilitation, error) {
419         var result []*models.Habilitation
420         loader := dataloaders.GetCourseLoader(ctx)
421         items, errs := loader.LoadAll(coursesIDs)
422         onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name"})
423         var isPublic bool
424         var err error
425         universities := make([]string, len(coursesIDs))
426         for i := range coursesIDs {
427                 if kv.IsKeyNotFoundError(errs[i]) {
428                         continue
429                 }
430                 if err == nil {
431                         err = errs[i]
432                 }
433                 if err == nil {
434                         universities[i] = items[i].UniversityID
435                 }
436         }
437         if err == nil {
438                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                    ↪  universities)
439         }
440         if err == nil && !isPublic {
441                 err = errors.New("Habilitations not found")
```

```
442         }
443         if err == nil {
444                 if onlySummary {
445                         for _, item := range items {
446                                 for _, Habilitation := range
                                    ↪  item.Habilitations {
447                                         result = append(result,
                                            ↪  &models.Habilitation{
448                                                 ID:   Habilitation.ID,
449                                                 Name: Habilitation.Name,
450                                         })
451                                 }
452                         }
453                 } else {
454                         var keys []string
455                         for _, item := range items {
456                                 for _, hab := range item.Habilitations {
457                                         keys = append(keys, hab.ID)
458                                 }
459                         }
460                         loader := dataloaders.GetHabilitationLoader(ctx)
461                         results, errors := loader.LoadAll(keys)
462                         for i := range keys {
463                                 if kv.IsKeyNotFoundError(errors[i]) {
464                                         continue
465                                 }
466                                 if err == nil {
467                                         err = errors[i]
468                                 }
469                                 if err == nil && results[i] != nil {
470                                         result = append(result,
                                            ↪  results[i])
471                                 }
472                         }
473                 }
474         }
475         return result, err
476 }
477
```

```
478 func (r queryResolver) HabilitationsByIDs(ctx context.Context, ids
    ↪  []string) ([]*models.Habilitation, error) {
479         var result []*models.Habilitation
480         var err error
481         loader := dataloaders.GetHabilitationLoader(ctx)
482         result, errs := loader.LoadAll(ids)
483         universities := make([]string, len(ids))
484         for i := range ids {
485                 if kv.IsKeyNotFoundError(errs[i]) {
486                         continue
487                 }
488                 if err == nil {
489                         err = errs[i]
490                 }
491                 if err == nil {
492                         universities[i] = result[i].UniversityID
493                 }
494         }
495         var isPublic bool
496         if err == nil {
497                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                    ↪  universities)
498         }
499         if err == nil && !isPublic {
500                 err = errors.New("Habilitations not found")
501         }
502         return result, err
503 }
504
505 func (r queryResolver) Habilitation(ctx context.Context, id string)
    ↪  (*models.Habilitation, error) {
506         loader := dataloaders.GetHabilitationLoader(ctx)
507         var isPublic bool
508         result, err := loader.Load(id)
509         if err == nil {
510                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  result.UniversityID)
511         }
512         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
```

```
513                 err = errors.New("Habilitation not found")
514         }
515         return result, err
516 }
517
518 func (r queryResolver) DisciplineOffer(ctx context.Context, id string)
    ↪  (*models.DisciplineOffer, error) {
519         loader := dataloaders.GetDisciplineOfferLoader(ctx)
520         result, err := loader.Load(id)
521         var isPublic bool
522         if err == nil && result != nil {
523                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  result.UniversityID)
524         }
525         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
526                 err = errors.New("Discipline Offer not found")
527         }
528         return result, err
529 }
530
531 func (r queryResolver) DisciplineOffersByCampus(ctx context.Context,
    ↪  campusID string) ([]*models.DisciplineOffer, error) {
532         var result []*models.DisciplineOffer
533         campusLoader := dataloaders.GetCampusLoader(ctx)
534         row, err := campusLoader.Load(campusID)
535         var isPublic bool
536         if err == nil && row != nil {
537                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  row.UniversityID)
538         }
539         if err == nil && !isPublic {
540                 err = errors.New("Discipline Offers not found")
541         }
542         if err == nil {
543                 result, err = loadDisciplineOffers(ctx,
                    ↪  row.DisciplineOffers)
544         }
545         return result, err
546 }
```

```
547
548 func (r queryResolver) DisciplineOffersByIDs(ctx context.Context, ids
    ↪  []string) ([]*models.DisciplineOffer, error) {
549         var result []*models.DisciplineOffer
550         var err error
551         loader := dataloaders.GetDisciplineOfferLoader(ctx)
552         result, errs := loader.LoadAll(ids)
553         universities := make([]string, len(ids))
554         for i := range ids {
555                 if kv.IsKeyNotFoundError(errs[i]) {
556                         continue
557                 }
558                 if err == nil {
559                         err = errs[i]
560                 }
561                 if err == nil {
562                         universities[i] = result[i].UniversityID
563                 }
564         }
565         var isPublic bool
566         if err == nil {
567                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                  ↪  universities)
568         }
569         if err == nil && !isPublic {
570                 err = errors.New("Discipline Offers not found")
571         }
572         return result, err
573 }
574
575 func (r queryResolver) Discipline(ctx context.Context, id string)
    ↪  (*models.Discipline, error) {
576         loader := dataloaders.GetDisciplineLoader(ctx)
577         result, err := loader.Load(id)
578         var isPublic bool
579         if err == nil && result != nil {
580                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                  ↪  result.UniversityID)
581         }
```

```
582         if kv.IsKeyNotFoundError(err) || err == nil && !isPublic {
583                 err = errors.New("Discipline not found")
584         }
585         return result, err
586 }
587
588 func (r queryResolver) DisciplinesByHabilitationAndStep(ctx
    ↪  context.Context, habilitationID string, stepID string)
    ↪  ([]*models.Discipline, error) {
589         var result []*models.Discipline
590         onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name"})
591         habilitationLoader := dataloaders.GetHabilitationLoader(ctx)
592         row, err := habilitationLoader.Load(habilitationID)
593         var isPublic bool
594         if err == nil && row != nil {
595                 isPublic, err = r.checkIfUniversityIsPublic(ctx,
                    ↪  row.UniversityID)
596         }
597         if err == nil && !isPublic {
598                 err = errors.New("Discipline not found")
599         }
600         if err == nil {
601                 if onlySummary {
602                         for _, stepDiscipline := range
                            ↪  row.StepDisciplines {
603                                 if row.Steps[stepDiscipline.StepID].ID ==
                                    ↪  stepID {
604                                         result = append(result,
                                            ↪  &models.Discipline{
605                                                 ID:
                                                    ↪  stepDiscipline.Discipline.ID,
606                                                 Name:
                                                    ↪  stepDiscipline.Discipline.Name,
607                                         })
608                                 }
609                         }
610                 } else {
611                         var ids []string
```

```
612                         for _, stepDiscipline := range
                            ↪  row.StepDisciplines {
613                                 if row.Steps[stepDiscipline.StepID].ID ==
                                    ↪  stepID {
614                                         ids = append(ids,
                                            ↪  stepDiscipline.Discipline.ID)
615                                 }
616                         }
617                         loader := dataloaders.GetDisciplineLoader(ctx)
618                         results, errors := loader.LoadAll(ids)
619                         for i := range ids {
620                                 if kv.IsKeyNotFoundError(errors[i]) {
621                                         continue
622                                 }
623                                 if err == nil {
624                                         err = errors[i]
625                                 }
626                                 if err == nil && results[i] != nil {
627                                         result = append(result,
                                            ↪  results[i])
628                                 }
629                         }
630                 }
631         }
632         return result, err
633 }
634
635 func (r queryResolver) DisciplinesByIDs(ctx context.Context, ids
    ↪  []string) ([]*models.Discipline, error) {
636         var result []*models.Discipline
637         var err error
638         loader := dataloaders.GetDisciplineLoader(ctx)
639         result, errs := loader.LoadAll(ids)
640         universities := make([]string, len(ids))
641         for i := range ids {
642                 if kv.IsKeyNotFoundError(errs[i]) {
643                         continue
644                 }
645                 if err == nil {
```

```
646                         err = errs[i]
647                 }
648                 if err == nil {
649                         universities[i] = result[i].UniversityID
650                 }
651         }
652         var isPublic bool
653         if err == nil {
654                 isPublic, err = r.checkIfUniversitiesArePublic(ctx,
                    ↪  universities)
655         }
656         if err == nil && !isPublic {
657                 err = errors.New("Disciplines not found")
658         }
659         return result, err
660 }
661
662 func (r queryResolver) Teams(ctx context.Context, disciplineID string)
    ↪  ([]*graphql.Team, error) {
663         var err error
664         var result []*graphql.Team
665         loader := dataloaders.GetDisciplineOfferLoader(ctx)
666         var discipline *models.DisciplineOffer
667         discipline, err = loader.Load(disciplineID)
668         if err == nil {
669                 result = make([]*graphql.Team, len(discipline.Teams))
670                 for i := range result {
671                         team := graphql.ToTeam(discipline, i)
672                         result[i] = &team
673                 }
674         }
675         return result, err
676 }
677
678 func (r queryResolver) Team(ctx context.Context, disciplineID, teamID
    ↪  string) (*graphql.Team, error) {
679         var err error
680         var result *graphql.Team
681         loader := dataloaders.GetDisciplineOfferLoader(ctx)
```

```
682         var discipline *models.DisciplineOffer
683         discipline, err = loader.Load(disciplineID)
684         if err == nil {
685                 for i, team := range discipline.Teams {
686                         if team.ID == teamID {
687                                 team := graphql.ToTeam(discipline, i)
688                                 result = &team
689                         }
690                 }
691                 if result == nil {
692                         err = errors.New("Team not found")
693                 }
694         }
695         return result, err
696 }
697
698 func (r queryResolver) SearchDisciplines(ctx context.Context,
    ↪  universityID string, query *graphql.SearchExpression, options
    ↪  *graphql.SearchOptions) (*graphql.DisciplineSearchResult, error) {
699         return searchDisciplines(ctx, r.handler, universityID, query,
           ↪  options)
700 }
701
702 func (r queryResolver) SearchDisciplineOffers(ctx context.Context,
    ↪  periodID string, query *graphql.SearchExpression, options
    ↪  *graphql.SearchOptions) (*graphql.DisciplineOfferSearchResult, error)
    ↪  {
703         return searchDisciplineOffers(ctx, r.handler, periodID, query,
           ↪  options)
704 }
705
706 func (r queryResolver) SearchTeams(ctx context.Context, periodID string,
    ↪  query *graphql.SearchExpression, options *graphql.SearchOptions)
    ↪  (*graphql.TeamSearchResult, error) {
707         return searchTeams(ctx, r.handler, periodID, query, options)
708 }
709
```

```
710 func (r queryResolver) SearchTeachers(ctx context.Context, periodID
    ↪  string, query *graphql.SearchExpression, options
    ↪  *graphql.SearchOptions) (*graphql.TeacherSearchResult, error) {
711         return searchTeachers(ctx, r.handler, periodID, query, options)
712 }
713
714 func (r queryResolver) Plan(ctx context.Context, id string) (*models.Plan,
    ↪  error) {
715         loader := dataloaders.GetPlanLoader(ctx)
716         result, err := loader.Load(id)
717         if err == nil && result != nil {
718                 authorized := result.Public
719                 if err == nil && !authorized {
720                         var userID string
721                         userID, err = r.getUserID(ctx)
722                         authorized = len(userID) > 0 && userID ==
                            ↪  result.User.ID
723                 }
724                 if err == nil && !authorized {
725                         err = errAccessDenied
726                 }
727         }
728         return result, err
729 }
730
731 func (r queryResolver) Plans(ctx context.Context) ([]*models.Plan, error)
    ↪  {
732         var result []*models.Plan
733         req := r.handler.GetRequest(ctx)
734         auth := r.handler.GetAuthBoss()
735         user, err := auth.CurrentUser(req)
736         if err == nil && len(user.GetPID()) == 0 {
737                 err = errAccessDenied
738         }
739         if err == nil {
740                 if appUser, ok := user.(*models.User); ok {
741                         onlySummary := answerWithSummaryOnly(ctx,
                            ↪  []string{"id", "name", "period", "university",
                            ↪  "createdAt", "lastModified"})
```

```
742                     result = make([]*models.Plan, len(appUser.Plans))
743                     if onlySummary {
744                             for i, plan := range appUser.Plans {
745                                     result[i] = &models.Plan{
746                                             ID:           plan.ID,
747                                             Name:         plan.Name,
748                                             Period:
                                                ↪  plan.Period,
749                                             University:
                                                ↪  plan.University,
750                                             CreatedAt:
                                                ↪  plan.CreatedAt,
751                                             LastModified:
                                                ↪  plan.LastModified,
752                                     }
753                             }
754                     } else {
755                             ids := make([]string, len(appUser.Plans))
756                             for i, plan := range appUser.Plans {
757                                     ids[i] = plan.ID
758                             }
759                             loader := dataloaders.GetPlanLoader(ctx)
760                             results, errors := loader.LoadAll(ids)
761                             result = result[:0]
762                             for i := range ids {
763                                     if
                                        ↪  kv.IsKeyNotFoundError(errors[i])
                                        ↪  {
764                                             continue
765                                     }
766                                     if err == nil {
767                                             err = errors[i]
768                                     }
769                                     if err == nil && results[i] !=
                                        ↪  nil {
770                                             result = append(result,
                                                ↪  results[i])
771                                     }
772                             }
```

```
773                         }
774                 } else {
775                         err = errors.New("Internal server error")
776                 }
777         }
778         return result, err
779 }
```

Arquivo resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5          "net/http"
6
7          "github.com/fjorgemota/gurudamatricula/util/clock"
8
9          "github.com/fjorgemota/gurudamatricula/graphql"
10         "github.com/fjorgemota/gurudamatricula/robot/importer"
11         "github.com/fjorgemota/gurudamatricula/util/indexer"
12         "github.com/volatiletech/authboss"
13
14         "github.com/fjorgemota/gurudamatricula/util/kv"
15 )
16
17 type Handler interface {
18         GetClock() clock.Clock
19         GetAuthBoss() *authboss.Authboss
20         GetRequest(context.Context) *http.Request
21         GetStore(context.Context) (kv.Store, error)
22         GetIndex(ctx context.Context, data importer.IndexData)
           ↪  (indexer.Index, error)
23 }
24
25 type Resolver struct {
26         Handler Handler
27 }
28
29 func (r Resolver) Campus() graphql.CampusResolver {
```

```
30          return campusResolver{handler: r.Handler}
31  }
32
33  func (r Resolver) Course() graphql.CourseResolver {
34          return courseResolver{handler: r.Handler}
35  }
36
37  func (r Resolver) CourseOffer() graphql.CourseOfferResolver {
38          return courseOfferResolver{handler: r.Handler}
39  }
40
41  func (r Resolver) Discipline() graphql.DisciplineResolver {
42          return disciplineResolver{handler: r.Handler}
43  }
44
45  func (r Resolver) DisciplineOffer() graphql.DisciplineOfferResolver {
46          return disciplineOfferResolver{handler: r.Handler}
47  }
48
49  func (r Resolver) Habilitation() graphql.HabilitationResolver {
50          return habilitationResolver{handler: r.Handler}
51  }
52
53  func (r Resolver) HourMinute() graphql.HourMinuteResolver {
54          return hourMinuteResolver{}
55  }
56
57  func (r Resolver) Mutation() graphql.MutationResolver {
58          return mutationResolver{handler: r.Handler}
59  }
60
61  func (r Resolver) Period() graphql.PeriodResolver {
62          return periodResolver{handler: r.Handler}
63  }
64
65  func (r Resolver) Plan() graphql.PlanResolver {
66          return planResolver{handler: r.Handler}
67  }
68
```

```
69 func (r Resolver) PlanDisciplineData() graphql.PlanDisciplineDataResolver
   ↪ {
70         return planDisciplineResolver{handler: r.Handler}
71 }
72
73 func (r Resolver) PlanDisciplineSummary()
   ↪  graphql.PlanDisciplineSummaryResolver {
74         return planDisciplineSummaryResolver{handler: r.Handler}
75 }
76
77 func (r Resolver) PlanDisciplineOfferData()
   ↪  graphql.PlanDisciplineOfferDataResolver {
78         return disciplineOfferResolver{handler: r.Handler}
79 }
80
81 func (r Resolver) PlanVersion() graphql.PlanVersionResolver {
82         return planVersionResolver{handler: r.Handler}
83 }
84
85 func (r Resolver) Query() graphql.QueryResolver {
86         return queryResolver{handler: r.Handler}
87 }
88
89 func (r Resolver) Room() graphql.RoomResolver {
90         return roomResolver{handler: r.Handler}
91 }
92
93 func (r Resolver) Step() graphql.StepResolver {
94         return stepResolver{handler: r.Handler}
95 }
96
97 func (r Resolver) Teacher() graphql.TeacherResolver {
98         return teacherResolver{handler: r.Handler}
99 }
100
101 func (r Resolver) Team() graphql.TeamResolver {
102         return teamResolver{handler: r.Handler}
103 }
104
```

```
105 func (r Resolver) University() graphql.UniversityResolver {
106         return universityResolver{handler: r.Handler}
107 }
108
109 func (r Resolver) UniversityData() graphql.UniversityDataResolver {
110         return universityDataResolver{handler: r.Handler}
111 }
112
113 func (r Resolver) User() graphql.UserResolver {
114         return userResolver{handler: r.Handler}
115 }
116
117 var _ graphql.ResolverRoot = Resolver{}
```

Arquivo room_resolver.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/graphql"
7          "github.com/fjorgemota/gurudamatricula/models"
8  )
9
10 type roomResolver struct {
11         handler Handler
12 }
13
14 func (r roomResolver) University(ctx context.Context, obj *models.Room)
   ↪  (*models.University, error) {
15         return loadUniversity(ctx, obj.UniversityID)
16 }
17
18 func (r roomResolver) Campus(ctx context.Context, obj *models.Room)
   ↪  (*models.Campus, error) {
19         return loadCampus(ctx, obj.Campus)
20 }
21
```

```
22 func (r roomResolver) Period(ctx context.Context, obj *models.Room)
   ↪ (*models.Period, error) {
23         return loadPeriod(ctx, obj.Period)
24 }
25
26 func (r roomResolver) SearchTeams(ctx context.Context, obj *models.Room,
   ↪ periodID *string, query *graphql.SearchExpression, options
   ↪ *graphql.SearchOptions) (*graphql.TeamSearchResult, error) {
27         if periodID == nil {
28                 periodID = &obj.Period.ID
29         }
30         newQuery := getNewQuery(query, "room_id", obj.ID)
31         return searchTeams(ctx, r.handler, *periodID, newQuery, options)
32 }
```

Arquivo search.go

```
1  package resolvers
2
3  import (
4          "context"
5
6          "github.com/fjorgemota/gurudamatricula/search"
7
8          "github.com/fjorgemota/gurudamatricula/util/kv"
9
10         "github.com/fjorgemota/gurudamatricula/graphql"
11         "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
12         "github.com/fjorgemota/gurudamatricula/models"
13         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
14         "github.com/fjorgemota/gurudamatricula/robot/importer"
15         "github.com/fjorgemota/gurudamatricula/util/indexer"
16 )
17
18 func getFilter(options *graphql.SearchOptions) graphql.SearchFilter {
19         filter := graphql.SearchFilterNone
20         if options != nil && options.Filter != nil {
21                 filter = *options.Filter
22         }
23         return filter
```

```
24 }
25
26 func getSearchOptions(options *graphql.SearchOptions)
   ↪ indexer.SearchOptions {
27         var opts indexer.SearchOptions
28         if options != nil {
29                 if options.After != nil {
30                         opts.After = uint32(*options.After)
31                 }
32                 if options.Before != nil {
33                         opts.Before = uint32(*options.Before)
34                 }
35                 if options.Limit != nil {
36                         opts.Limit = uint32(*options.Limit)
37                 }
38                 if options.Version != nil {
39                         opts.Version = *options.Version
40                 }
41         }
42         if opts.Limit == 0 {
43                 opts.Limit = 10
44         }
45         if opts.Limit > 100 {
46                 opts.Limit = 100
47         }
48         return opts
49 }
50
51 func getAllowedDisciplines(ctx context.Context, handler Handler, index
   ↪ indexer.Index, filter graphql.SearchFilter, field, version string)
   ↪ (indexer.Expression, string, error) {
52         var err error
53         var result indexer.Expression = indexer.All{}
54         req := handler.GetRequest(ctx)
55         auth := handler.GetAuthBoss()
56         currentUser, err := auth.CurrentUser(req)
57         if err == nil && currentUser != nil {
58                 user := currentUser.(*models.User)
```

```
59                 result, version, err = search.GetAllowedDisciplines(index,
                   ↪  search.Options{
60                         Filter:               filter,
61                         Courses:              user.Courses,
62                         CompletedDisciplines: user.CompletedDisciplines,
63                         Field:                field,
64                         Version:              version,
65                 })
66         }
67         return result, version, err
68 }
69
70 func searchDisciplines(ctx context.Context, handler Handler, universityID
   ↪  string, query *graphql.SearchExpression, options
   ↪  *graphql.SearchOptions) (*graphql.DisciplineSearchResult, error) {
71         var result graphql.DisciplineSearchResult
72         index, err := handler.GetIndex(ctx, importer.IndexData{
73                 Target:   ddiffer.TargetDiscipline,
74                 ParentID: universityID,
75         })
76         var expression indexer.Expression
77         if err == nil {
78                 expression, err = toExpression(query)
79         }
80         opts := getSearchOptions(options)
81         filter := getFilter(options)
82         if err == nil && filter != graphql.SearchFilterNone {
83                 var result indexer.Expression
84                 result, opts.Version, err = getAllowedDisciplines(ctx,
                   ↪  handler, index, filter, "generic_id", opts.Version)
85                 if err == nil {
86                         expression = indexer.And{
87                                 Operations: []indexer.Expression{result,
                                   ↪  expression},
88                         }
89                 }
90         }
91         if err == nil {
```

```
92                  onlySummary := answerWithSummaryOnly(ctx, []string{"id",
                    ↪  "generic_id", "code", "version", "name", "step",
                    ↪  "habilitation", "type", "description", "course",
                    ↪  "related"})
93                  it := index.SearchWithOptions(expression, opts)
94                  result.Results = make([]*models.Discipline, 0,
                    ↪  opts.Limit)
95                  info := it.Info()
96                  result.Info = &graphql.SearchResultInfo{
97                          Page: &graphql.SearchPageInfo{
98                                  After:  int(info.Page.After),
99                                  Before: int(info.Page.Before),
100                         },
101                         Query: &graphql.SearchQueryInfo{
102                                 Count:   int(info.Query.Count),
103                                 First:   int(info.Query.First),
104                                 Last:    int(info.Query.Last),
105                                 Version: info.Query.Version,
106                         },
107                 }
108                 var ids []string
109                 for err == nil {
110                         var ref models.BasicDiscipline
111                         err = it.Next(&ref)
112                         if err == nil {
113                                 if onlySummary {
114                                         row := models.Discipline{
115                                                 ID:           ref.ID,
116                                                 GenericID:
                                                    ↪  ref.GenericID,
117                                                 Code:         ref.Code,
118                                                 Version:
                                                    ↪  ref.Version,
119                                                 Name:         ref.Name,
120                                                 Type:         ref.Type,
121                                                 Description:
                                                    ↪  ref.Description,
122                                                 Course:       ref.Course,
```

```
123                                             Habilitation:
                                                ↪ ref.Habilitation,
124                                             Related:
                                                ↪ ref.Related,
125                                             RelatedItems:
                                                ↪ ref.RelatedItems,
126                                     }
127                                     result.Results =
                                        ↪ append(result.Results, &row)
128                             } else {
129                                     ids = append(ids, ref.ID)
130                             }
131                     }
132             }
133             if indexer.IsDoneError(err) {
134                     err = nil
135             }
136             if err == nil && !onlySummary {
137                     loader := dataloaders.GetDisciplineLoader(ctx)
138                     rows, errors := loader.LoadAll(ids)
139                     for i, row := range rows {
140                             if kv.IsKeyNotFoundError(errors[i]) {
141                                     continue
142                             }
143                             if err == nil {
144                                     err = errors[i]
145                             }
146                             if err == nil && row != nil {
147                                     result.Results =
                                        ↪ append(result.Results, row)
148                             }
149                     }
150             }
151     }
152     return &result, err
153 }
154
155 func getSearchInfo(it indexer.IndexIterator) *graphql.SearchResultInfo {
156     info := it.Info()
```

```
157         return &graphql.SearchResultInfo{
158                 Page: &graphql.SearchPageInfo{
159                         After:  int(info.Page.After),
160                         Before: int(info.Page.Before),
161                 },
162                 Query: &graphql.SearchQueryInfo{
163                         Count:   int(info.Query.Count),
164                         First:   int(info.Query.First),
165                         Last:    int(info.Query.Last),
166                         Version: info.Query.Version,
167                 },
168         }
169 }
170
171 func searchDisciplineOffers(ctx context.Context, handler Handler,
    ↪  periodID string, query *graphql.SearchExpression, options
    ↪  *graphql.SearchOptions) (*graphql.DisciplineOfferSearchResult, error)
    ↪  {
172         var result graphql.DisciplineOfferSearchResult
173         index, err := handler.GetIndex(ctx, importer.IndexData{
174                 ParentID: periodID,
175                 Target:   ddiffer.TargetDisciplineOffer,
176         })
177         var expression indexer.Expression
178         if err == nil {
179                 expression, err = toExpression(query)
180         }
181         opts := getSearchOptions(options)
182         filter := getFilter(options)
183         if err == nil && filter != graphql.SearchFilterNone {
184                 var result indexer.Expression
185                 periodLoader := dataloaders.GetPeriodLoader(ctx)
186                 var period *models.Period
187                 period, err = periodLoader.Load(periodID)
188                 if err == nil {
189                         var disciplines indexer.Index
190                         disciplines, err = handler.GetIndex(ctx,
                        ↪  importer.IndexData{
191                                 ParentID: period.UniversityID,
```

```
192                             Target:   ddiffer.TargetDiscipline,
193                     })
194                     result, _, err = getAllowedDisciplines(ctx,
                        ↪  handler, disciplines, filter, "generic_id",
                        ↪  "latest")
195             }
196             if err == nil {
197                     expression = indexer.And{
198                             Operations: []indexer.Expression{result,
                                ↪  expression},
199                     }
200             }
201     }
202     if err == nil {
203             onlySummary := answerWithSummaryOnly(ctx, []string{"id",
                ↪  "code", "version", "name", "period", "campus"})
204             it := index.SearchWithOptions(expression, opts)
205             result.Results = make([]*models.DisciplineOffer, 0,
                ↪  opts.Limit)
206             result.Info = getSearchInfo(it)
207             var ids []string
208             for err == nil {
209                     var ref models.BasicDisciplineOffer
210                     err = it.Next(&ref)
211                     if err == nil {
212                             if onlySummary {
213                                     row := models.DisciplineOffer{
214                                             ID:      ref.ID,
215                                             Code:    ref.Code,
216                                             Version: ref.Version,
217                                             Name:    ref.Name,
218                                             Period:  ref.Period,
219                                             Campus:  ref.Campus,
220                                     }
221                                     result.Results =
                                        ↪  append(result.Results, &row)
222                             } else {
223                                     ids = append(ids, ref.ID)
224                             }
```

```
225                         }
226                 }
227                 if indexer.IsDoneError(err) {
228                         err = nil
229                 }
230                 if err == nil && !onlySummary {
231                         loader :=
                        ↪  dataloaders.GetDisciplineOfferLoader(ctx)
232                         rows, errors := loader.LoadAll(ids)
233                         for i, row := range rows {
234                                 if kv.IsKeyNotFoundError(errors[i]) {
235                                         continue
236                                 }
237                                 if err == nil {
238                                         err = errors[i]
239                                 }
240                                 if err == nil && row != nil {
241                                         result.Results =
                                        ↪  append(result.Results, row)
242                                 }
243                         }
244                 }
245         }
246         return &result, err
247 }
248
249 func searchTeams(ctx context.Context, handler Handler, periodID string,
    ↪  query *graphql.SearchExpression, options *graphql.SearchOptions)
    ↪  (*graphql.TeamSearchResult, error) {
250         var result graphql.TeamSearchResult
251         index, err := handler.GetIndex(ctx, importer.IndexData{
252                 ParentID: periodID,
253                 Target:   ddiffer.TargetTeam,
254         })
255         var expression indexer.Expression
256         if err == nil {
257                 expression, err = toExpression(query)
258         }
259         opts := getSearchOptions(options)
```

```
260         filter := getFilter(options)
261         if err == nil && filter != graphql.SearchFilterNone {
262                 var result indexer.Expression
263                 periodLoader := dataloaders.GetPeriodLoader(ctx)
264                 var period *models.Period
265                 period, err = periodLoader.Load(periodID)
266                 if err == nil {
267                         var disciplines indexer.Index
268                         disciplines, err = handler.GetIndex(ctx,
                        ↪  importer.IndexData{
269                                 ParentID: period.UniversityID,
270                                 Target:   ddiffer.TargetDiscipline,
271                         })
272                         result, _, err = getAllowedDisciplines(ctx,
                        ↪  handler, disciplines, filter,
                        ↪  "discipline_generic_id", "latest")
273                 }
274                 if err == nil {
275                         expression = indexer.And{
276                                 Operations: []indexer.Expression{result,
                                ↪  expression},
277                         }
278                 }
279         }
280         if err == nil {
281                 onlySummary := answerWithSummaryOnly(ctx, []string{"id",
                ↪  "code", "version"})
282                 it := index.SearchWithOptions(expression, opts)
283                 result.Results = make([]*graphql.Team, 0, opts.Limit)
284                 result.Info = getSearchInfo(it)
285                 var refs []models.Team
286                 for err == nil {
287                         var ref models.Team
288                         err = it.Next(&ref)
289                         if err == nil {
290                                 if onlySummary {
291                                         row := graphql.Team{
292                                                 ID:      ref.ID,
293                                                 Code:    ref.Code,
```

```
294                                                     Version: ref.Version,
295                                             }
296                                             result.Results =
                                                ↪  append(result.Results, &row)
297                                     } else {
298                                             refs = append(refs, ref)
299                                     }
300                             }
301                     }
302                     if indexer.IsDoneError(err) {
303                             err = nil
304                     }
305                     if err == nil && !onlySummary {
306                             loader := dataloaders.GetTeamLoader(ctx)
307                             rows, errors := loader.LoadAll(refs)
308                             for i, row := range rows {
309                                     if err == nil {
310                                             err = errors[i]
311                                     }
312                                     if err == nil && row != nil {
313                                             result.Results =
                                                ↪  append(result.Results, row)
314                                     }
315                             }
316                     }
317             }
318             return &result, err
319     }
320
321     func searchTeachers(ctx context.Context, handler Handler, periodID string,
        ↪  query *graphql.SearchExpression, options *graphql.SearchOptions)
        ↪  (*graphql.TeacherSearchResult, error) {
322             var result graphql.TeacherSearchResult
323             index, err := handler.GetIndex(ctx, importer.IndexData{
324                     ParentID: periodID,
325                     Target:   ddiffer.TargetTeacher,
326             })
327             var expression indexer.Expression
328             if err == nil {
```

```
329                 expression, err = toExpression(query)
330         }
331         opts := getSearchOptions(options)
332         if err == nil {
333                 it := index.SearchWithOptions(expression, opts)
334                 result.Results = make([]*models.Teacher, 0, opts.Limit)
335                 result.Info = getSearchInfo(it)
336                 for err == nil {
337                         var ref models.Teacher
338                         err = it.Next(&ref)
339                         if err == nil {
340                                 result.Results = append(result.Results,
                                ↪  &ref)
341                         }
342                 }
343                 if indexer.IsDoneError(err) {
344                         err = nil
345                 }
346         }
347         return &result, err
348 }
349
350 func getNewQuery(query *graphql.SearchExpression, fieldName, value
    ↪  string) *graphql.SearchExpression {
351         newQuery := &graphql.SearchExpression{
352                 Type: graphql.SearchOperatorAnd,
353                 Expressions: []*graphql.SearchExpression{
354                         &graphql.SearchExpression{
355                                 Type:
                                ↪  graphql.SearchOperatorFieldTerminal,
356                                 Attribute: &fieldName,
357                                 Value:     &value,
358                         },
359                 },
360         }
361         if query != nil {
362                 newQuery.Expressions = append(newQuery.Expressions,
                ↪  query)
363         }
```

```
364         return newQuery
365 }
```

Arquivo search_expression.go

```
1  package resolvers
2
3  import (
4          "errors"
5
6          "github.com/fjorgemota/gurudamatricula/graphql"
7          "github.com/fjorgemota/gurudamatricula/util/indexer"
8  )
9
10 type queueExpressionItem struct {
11         parent     indexer.Expression
12         expression *graphql.SearchExpression
13 }
14
15 func getSearchExpression(expression *graphql.SearchExpression)
   ↪  (indexer.Expression, error) {
16         var result indexer.Expression
17         var err error
18         switch expression.Type {
19         case graphql.SearchOperatorAll:
20                 result = &indexer.All{}
21         case graphql.SearchOperatorAnd:
22                 result = &indexer.And{}
23         case graphql.SearchOperatorOr:
24                 result = &indexer.Or{}
25         case graphql.SearchOperatorNot:
26                 result = &indexer.Not{}
27         case graphql.SearchOperatorTerminal:
28                 result = &indexer.Terminal{
29                         Value: *expression.Value,
30                 }
31         case graphql.SearchOperatorFieldTerminal:
32                 result = &indexer.FieldTerminal{
33                         Attribute: *expression.Attribute,
34                         Value:     *expression.Value,
```

```
35                  }
36          }
37          if result == nil {
38                  err = errors.New("Unrecognized expression")
39          }
40          return result, err
41  }
42
43  func addExpressionQueue(queue []queueExpressionItem, itemResult
    ↪  indexer.Expression, item queueExpressionItem) ([]queueExpressionItem,
    ↪  error) {
44          var err error
45          switch item.expression.Type {
46          case graphql.SearchOperatorNot:
47                  if len(item.expression.Expressions) != 1 {
48                          err = errors.New("Need to have exactly 1
                            ↪  expression as child of a expression Not")
49                  }
50                  fallthrough
51          case graphql.SearchOperatorAnd:
52                  fallthrough
53          case graphql.SearchOperatorOr:
54                  for _, child := range item.expression.Expressions {
55                          queue = append(queue, queueExpressionItem{
56                                  parent:     itemResult,
57                                  expression: child,
58                          })
59                  }
60          }
61          return queue, err
62  }
63
64  func addToParentExpression(parent indexer.Expression, itemResult
    ↪  indexer.Expression) {
65          switch v := parent.(type) {
66          case *indexer.And:
67                  v.Operations = append(v.Operations, itemResult)
68          case *indexer.Or:
69                  v.Operations = append(v.Operations, itemResult)
```

```
70          case *indexer.Not:
71                  v.Operation = itemResult
72          }
73  }
74
75  func toExpression(expression *graphql.SearchExpression)
    ↪  (indexer.Expression, error) {
76          var result indexer.Expression
77          result = indexer.All{}
78          var err error
79          queue := []queueExpressionItem{}
80          if expression != nil {
81                  queue = append(queue, queueExpressionItem{
82                          parent:     nil,
83                          expression: expression,
84                  })
85          }
86          for len(queue) > 0 && err == nil {
87                  var item queueExpressionItem
88                  var itemResult indexer.Expression
89                  item, queue = queue[0], queue[1:]
90                  itemResult, err = getSearchExpression(item.expression)
91                  if item.parent == nil {
92                          result = itemResult
93                  } else {
94                          addToParentExpression(item.parent, itemResult)
95                  }
96                  if err == nil {
97                          queue, err = addExpressionQueue(queue, itemResult,
                            ↪  item)
98                  }
99          }
100         return result, err
101 }
```

Arquivo table_resolver.go

```
1   package resolvers
2
3   import (
```

```
4         "github.com/fjorgemota/gurudamatricula/graphql"
5         "github.com/fjorgemota/gurudamatricula/models"
6 )
7
8 func resolveTables(tables []models.Table, columns []models.TableColumn,
  ↪  rows []models.TableRow) ([]*graphql.Table, error) {
9         result := make([]*graphql.Table, len(tables))
10        for tableID, table := range tables {
11                tableResult := graphql.Table{
12                        ID:          table.ID,
13                        Title:       table.Title,
14                        Description: &table.Description,
15                }
16                for _, column := range columns {
17                        if column.TableID == tableID {
18                                localColumn := column
19                                tableResult.Columns =
                                  ↪  append(tableResult.Columns,
                                  ↪  &localColumn)
20                        }
21                }
22                for _, row := range rows {
23                        if columns[row.Column].TableID == tableID {
24                                localRow := row
25                                tableResult.Rows =
                                  ↪  append(tableResult.Rows, &localRow)
26                        }
27                }
28                result[tableID] = &tableResult
29        }
30        return result, nil
31 }
```

Arquivo teacher_resolver.go

```
1 package resolvers
2
3 import (
4         "context"
5
```

```
6         "github.com/fjorgemota/gurudamatricula/graphql"
7         "github.com/fjorgemota/gurudamatricula/models"
8 )
9
10 type teacherResolver struct {
11         handler Handler
12 }
13
14 func (r teacherResolver) University(ctx context.Context, obj
   ↪  *models.Teacher) (*models.University, error) {
15         return loadUniversity(ctx, obj.UniversityID)
16 }
17
18 func (r teacherResolver) Period(ctx context.Context, obj *models.Teacher)
   ↪  (*models.Period, error) {
19         return loadPeriod(ctx, obj.Period)
20 }
21
22 func (r teacherResolver) SearchTeams(ctx context.Context, obj
   ↪  *models.Teacher, periodID *string, query *graphql.SearchExpression,
   ↪  options *graphql.SearchOptions) (*graphql.TeamSearchResult, error) {
23         newQuery := getNewQuery(query, "teacher_period_id", obj.ID)
24         if periodID == nil {
25                 periodID = &obj.Period.ID
26         }
27         return searchTeams(ctx, r.handler, *periodID, newQuery, options)
28 }
```

Arquivo team_resolver.go

```
1 package resolvers
2
3 import (
4         "context"
5
6         "github.com/fjorgemota/gurudamatricula/graphql"
7         "github.com/fjorgemota/gurudamatricula/models"
8 )
9
10 type teamResolver struct {
```

```
11          handler Handler
12  }
13
14  func (r teamResolver) Campus(ctx context.Context, obj *graphql.Team)
    ↪  (*models.Campus, error) {
15          return loadCampus(ctx, models.ForeignKey{
16                  ID:   obj.Campus.ID,
17                  Name: obj.Campus.Name,
18          })
19  }
20
21  func (r teamResolver) Period(ctx context.Context, obj *graphql.Team)
    ↪  (*models.Period, error) {
22          return loadPeriod(ctx, models.ForeignKey{
23                  ID:   obj.Period.ID,
24                  Name: obj.Period.Name,
25          })
26  }
27
28  func (r teamResolver) University(ctx context.Context, obj *graphql.Team)
    ↪  (*models.University, error) {
29          return loadUniversity(ctx, obj.University.ID)
30  }
```

Arquivo university_resolver.go

```
1   package resolvers
2
3   import (
4           "context"
5
6           "github.com/fjorgemota/gurudamatricula/models/keygen"
7
8           "github.com/fjorgemota/gurudamatricula/models"
9   )
10
11  type universityResolver struct {
12          handler Handler
13  }
14
```

```
15 func (r universityResolver) Periods(ctx context.Context, obj
   ↪ *models.University) ([]*models.Period, error) {
16         return loadPeriods(ctx, obj.Periods)
17 }
18
19 func (r universityResolver) Courses(ctx context.Context, obj
   ↪ *models.University) ([]*models.Course, error) {
20         return loadCourses(ctx, obj.Courses)
21 }
22
23 type universityDataResolver struct {
24         handler Handler
25 }
26
27 func (r universityDataResolver) checkAuth(ctx context.Context, obj
   ↪ *models.UniversityData) error {
28         req := r.handler.GetRequest(ctx)
29         auth := r.handler.GetAuthBoss()
30         userID, err := auth.CurrentUserID(req)
31         if err == nil {
32                 userID, err = keygen.GenerateUserID(userID)
33         }
34         if len(userID) == 0 || userID != obj.UserID {
35                 err = errAccessDenied
36         }
37         return err
38 }
39
40 func (r universityDataResolver) processResponse(ctx context.Context, obj
   ↪ *models.UniversityData, value string) (string, error) {
41         var result string
42         err := r.checkAuth(ctx, obj)
43         if err == nil {
44                 result = value
45         }
46         return result, err
47 }
48
```

```
func (r universityDataResolver) QueueSize(ctx context.Context, obj
↪  *models.UniversityData) (int, error) {
        var result int
        err := r.checkAuth(ctx, obj)
        if err == nil {
                result = len(obj.Queue)
        }
        return result, nil
}

func (r universityDataResolver) BaseURL(ctx context.Context, obj
↪  *models.UniversityData) (string, error) {
        return r.processResponse(ctx, obj, obj.BaseURL)
}

func (r universityDataResolver) DeleteURL(ctx context.Context, obj
↪  *models.UniversityData) (string, error) {
        return r.processResponse(ctx, obj, obj.DeleteURL)
}

func (r universityDataResolver) Secret(ctx context.Context, obj
↪  *models.UniversityData) (string, error) {
        secret, err := r.processResponse(ctx, obj, obj.Secret)
        if err == nil && len(secret) > 0 {
                if secret[0] == '$' {
                        secret = ""
                        err = errAccessDenied
                } else {
                        secret = secret[1:]
                }
        }
        return secret, err
}
```

Arquivo user_resolver.go

```
package resolvers

import (
        "context"
```

```
5
6           "github.com/fjorgemota/gurudamatricula/graphql/dataloaders"
7
8           "github.com/fjorgemota/gurudamatricula/models"
9   )
10
11  type userResolver struct {
12          handler Handler
13  }
14
15  func (r userResolver) verifyAuth(ctx context.Context, obj *models.User)
    ↪  error {
16          auth := r.handler.GetAuthBoss()
17          req := r.handler.GetRequest(ctx)
18          userID, err := auth.CurrentUserID(req)
19          if err == nil && userID != obj.ID {
20                  err = errAccessDenied
21          }
22          return err
23  }
24
25  func (r userResolver) Courses(ctx context.Context, obj *models.User)
    ↪  ([]*models.Course, error) {
26          return loadCourses(ctx, obj.Courses)
27  }
28
29  func (r userResolver) Habilitations(ctx context.Context, obj
    ↪  *models.User) ([]*models.Habilitation, error) {
30          return loadHabilitations(ctx, obj.Habilitations)
31  }
32
33  func (r userResolver) Plans(ctx context.Context, obj *models.User)
    ↪  ([]*models.Plan, error) {
34          err := r.verifyAuth(ctx, obj)
35          onlySummary := answerWithSummaryOnly(ctx, []string{"id",
            ↪  "university", "period", "created_at", "last_modified",
            ↪  "name"})
36          results := make([]*models.Plan, 0, len(obj.Plans))
37          if onlySummary {
```

```
38                  for _, plan := range obj.Plans {
39                          results = append(results, &models.Plan{
40                                  ID:           plan.ID,
41                                  Name:         plan.Name,
42                                  CreatedAt:    plan.CreatedAt,
43                                  LastModified: plan.LastModified,
44                                  University:   plan.University,
45                                  Period:       plan.Period,
46                          })
47                  }
48          } else {
49                  ids := make([]string, len(obj.Plans))
50                  for i, plan := range obj.Plans {
51                          ids[i] = plan.ID
52                  }
53                  loader := dataloaders.GetPlanLoader(ctx)
54                  plans, errors := loader.LoadAll(ids)
55                  for i := range ids {
56                          if err == nil {
57                                  err = errors[i]
58                          }
59                          if err == nil && plans[i] != nil {
60                                  results[i] = plans[i]
61                          }
62                  }
63          }
64          return results, err
65  }
66  func (r userResolver) University(ctx context.Context, obj *models.User)
    ↪  (*models.University, error) {
67          err := r.verifyAuth(ctx, obj)
68          onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name"})
69          var result *models.University
70          if obj.University.ID != "" {
71                  if onlySummary {
72                          result = &models.University{
73                                  ID:   obj.University.ID,
74                                  Name: obj.University.Name,
75                          }
```

```
76                  } else {
77                          result, err = loadUniversity(ctx,
                            ↪  obj.University.ID)
78                  }
79          }
80          return result, err
81  }
82
83  func (r userResolver) Universities(ctx context.Context, obj *models.User)
    ↪  ([]*models.University, error) {
84          err := r.verifyAuth(ctx, obj)
85          onlySummary := answerWithSummaryOnly(ctx, []string{"id", "name",
            ↪  "acronym", "public"})
86          results := make([]*models.University, len(obj.Universities))
87          if onlySummary {
88                  for _, university := range obj.Universities {
89                          results = append(results, &models.University{
90                                  ID:      university.ID,
91                                  Name:    university.Name,
92                                  Acronym: university.Acronym,
93                                  Public:  university.Public,
94                          })
95                  }
96          } else {
97                  ids := make([]string, len(obj.Universities))
98                  for i, university := range obj.Universities {
99                          ids[i] = university.ID
100                 }
101                 loader := dataloaders.GetUniversityLoader(ctx)
102                 universities, errors := loader.LoadAll(ids)
103                 for i := range ids {
104                         if err == nil {
105                                 err = errors[i]
106                         }
107                         if err == nil && universities[i] != nil {
108                                 results[i] = universities[i]
109                         }
110                 }
111         }
```

```
112         return results, err
113 }
```

B.1.9 Pasta models/keygen

Arquivo base.go

```
1  package keygen
2
3  import (
4          "encoding/hex"
5          "errors"
6          "io"
7
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9  )
10
11 var errEmpty = errors.New("Value cannot be empty")
12 var errInvalidSize = errors.New("keys size is different of values size")
13
14 func genHash(dataset string, keys []string, values map[string]string)
   ↪  (string, error) {
15         err := errInvalidSize
16         if len(keys) == len(values) {
17                 err = nil
18         }
19         buf := pool.GetSha1()
20         defer pool.PutSha1(buf)
21         for _, key := range keys {
22                 value := values[key]
23                 if err == nil {
24                         _, err = io.WriteString(buf, key)
25                 }
26                 if err == nil {
27                         _, err = buf.Write([]byte("-"))
28                 }
29                 if err == nil && len(value) == 0 {
30                         err = newErrEmpty(dataset, key)
31                 }
32                 if err == nil {
```

```
33                          _, err = io.WriteString(buf, value)
34                  }
35                  if err == nil {
36                          _, err = buf.Write([]byte("|"))
37                  }
38          }
39          var hash string
40          if err == nil {
41                  var bs [20]byte
42                  hash = "gdm-" + dataset + "-" +
                    ↪  hex.EncodeToString(buf.Sum(bs[:0]))
43          }
44          return hash, err
45  }
```

Arquivo campus.go

```
1   package keygen
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/robot/format"
5   )
6
7   func GenerateCampusID(universityID string, period format.Period, campus
    ↪  format.Campus) (string, error) {
8           return genHash("campus", []string{"university", "period",
            ↪  "campus"}, map[string]string{
9                   "university": universityID,
10                  "period":     period.ID,
11                  "campus":     campus.ID,
12          })
13  }
```

Arquivo course.go

```
1   package keygen
2
3   import "github.com/fjorgemota/gurudamatricula/robot/format"
4
```

```
5  func GenerateCourseID(universityID string, course format.Course) (string,
   ↪  error) {
6          return genHash("course", []string{"university", "course"},
           ↪  map[string]string{
7                  "university": universityID,
8                  "course":     course.ID,
9          })
10 }
```

Arquivo discipline.go

```
1  package keygen
2
3  import "github.com/fjorgemota/gurudamatricula/robot/format"
4
5  func GenerateDisciplineID(universityID string, discipline
   ↪  format.Discipline) (string, error) {
6          return genHash("discipline", []string{"university", "discipline"},
           ↪  map[string]string{
7                  "university": universityID,
8                  "discipline": discipline.ID,
9          })
10 }
```

Arquivo discipline_generic.go

```
1  package keygen
2
3  func GenerateDisciplineGenericID(universityID, id string) (string, error)
   ↪  {
4          return genHash("generic-discipline", []string{"university",
           ↪  "generic discipline"}, map[string]string{
5                  "university":         universityID,
6                  "generic discipline": id,
7          })
8  }
```

Arquivo discipline_offer.go

```go
package keygen

import (
        "github.com/fjorgemota/gurudamatricula/robot/format"
)

func GenerateDisciplineOfferID(universityID string, period format.Period,
    ↪  campus format.Campus, offer format.DisciplineOffer) (string, error) {
        return genHash("discipline-offer", []string{"university",
        ↪  "period", "campus", "discipline offer"}, map[string]string{
                "university":       universityID,
                "period":           period.ID,
                "campus":           campus.ID,
                "discipline offer": offer.ID,
        })
}
```

Arquivo errors.go

```go
package keygen

import (
        "fmt"

        "github.com/pkg/errors"
)

type errValueDataset struct {
        dataset string
        key     string
        parent  error
}

func (err errValueDataset) Cause() error {
        return err.parent
}

func (err errValueDataset) Error() string {
        return fmt.Sprintf("Error on key '%s' on dataset '%s': %s",
        ↪  err.key, err.dataset, err.parent.Error())
```

```
21 }
22
23 func newErrEmpty(dataset, key string) error {
24         return errValueDataset{
25                 dataset: dataset,
26                 key:     key,
27                 parent:  errEmpty,
28         }
29 }
30
31 func IsErrEmpty(err error) bool {
32         return errors.Cause(err) == errEmpty
33 }
```

Arquivo habilitation.go

```
1  package keygen
2
3  import "github.com/fjorgemota/gurudamatricula/robot/format"
4
5  func GenerateHabilitationID(universityID string, habilitation
   ↪  format.Habilitation) (string, error) {
6          return genHash("habilitation", []string{"university", "course",
           ↪  "habilitation"}, map[string]string{
7                  "university":   universityID,
8                  "course":       habilitation.Course.ID,
9                  "habilitation": habilitation.ID,
10         })
11 }
```

Arquivo period.go

```
1  package keygen
2
3  import "github.com/fjorgemota/gurudamatricula/robot/format"
4
5  func GeneratePeriodID(universityID string, period format.Period) (string,
   ↪  error) {
6          return genHash("period", []string{"university", "period"},
           ↪  map[string]string{
```

```
                "university": universityID,
                "period":     period.ID,
        })
}
```

Arquivo room.go

```
package keygen

import "github.com/fjorgemota/gurudamatricula/robot/format"

func GenerateRoomID(universityID string, campus format.Campus, room
↪  format.Room) (string, error) {
        return genHash("room", []string{"university", "campus", "room"},
        ↪  map[string]string{
                "university": universityID,
                "campus":     campus.ID,
                "room":       room.ID,
        })
}
```

Arquivo room_period.go

```
package keygen

import "github.com/fjorgemota/gurudamatricula/robot/format"

func GenerateRoomByPeriodID(universityID string, period format.Period,
↪  campus format.Campus, room format.Room) (string, error) {
        return genHash("room-period", []string{"university", "period",
        ↪  "campus", "room"}, map[string]string{
                "university": universityID,
                "period":     period.ID,
                "campus":     campus.ID,
                "room":       room.ID,
        })
}
```

Arquivo step.go

```
package keygen

import "github.com/fjorgemota/gurudamatricula/robot/format"

func GenerateStepID(universityID string, habilitation format.Habilitation,
↪  step format.Step) (string, error) {
        return genHash("habilitation-step", []string{"university",
        ↪  "course", "habilitation", "step"}, map[string]string{
                "university":   universityID,
                "course":       habilitation.Course.ID,
                "habilitation": habilitation.ID,
                "step":         step.ID,
        })
}
```

Arquivo table.go

```
package keygen

import "github.com/fjorgemota/gurudamatricula/robot/format"

func GenerateTableID(universityID, baseID string, table format.Table)
↪  (string, error) {
        return genHash("table", []string{"university", "base-id",
        ↪  "table"}, map[string]string{
                "university": universityID,
                "base-id":    baseID,
                "table":      table.ID,
        })
}
```

Arquivo teacher.go

```
package keygen

import "github.com/fjorgemota/gurudamatricula/robot/format"

func GenerateTeacherID(universityID string, teacher format.Teacher)
↪  (string, error) {
```

```
6         return genHash("teacher", []string{"university", "teacher"},
          ↪  map[string]string{
7                 "university": universityID,
8                 "teacher":    teacher.ID,
9         })
10 }
```

Arquivo teacher_period.go

```
1  package keygen
2
3  import "github.com/fjorgemota/gurudamatricula/robot/format"
4
5  func GenerateTeacherByPeriodID(universityID string, period format.Period,
   ↪  teacher format.Teacher) (string, error) {
6         return genHash("teacher-period", []string{"university", "period",
          ↪  "teacher"}, map[string]string{
7                 "university": universityID,
8                 "period":     period.ID,
9                 "teacher":    teacher.ID,
10        })
11 }
```

Arquivo team.go

```
1  package keygen
2
3  import (
4         "github.com/fjorgemota/gurudamatricula/robot/format"
5  )
6
7  func GenerateTeamID(universityID string, team format.Team) (string,
   ↪  error) {
8         return genHash("team", []string{"university", "period", "campus",
          ↪  "discipline offer", "team"}, map[string]string{
9                 "university":       universityID,
10                "period":           team.Period.ID,
11                "campus":           team.Campus.ID,
12                "discipline offer": team.Discipline.ID,
```

```
13                  "team":             team.ID,
14          })
15  }
```

Arquivo user.go

```
1   package keygen
2
3   func GenerateUserID(userPID string) (string, error) {
4           return genHash("user", []string{"user-pid"}, map[string]string{
5                   "user-pid": userPID,
6           })
7   }
8
9   func GenerateUserSelector(selectorType, selectorValue string) (string,
    ↪  error) {
10          return genHash("selector", []string{"type", "value"},
            ↪  map[string]string{
11                  "type":  selectorType,
12                  "value": selectorValue,
13          })
14  }
```

B.1.10 Pasta robot/ddiffer

Arquivo base.go

```
1   package ddiffer
2
3   import (
4           "encoding/gob"
5           "errors"
6           "io"
7
8           "github.com/fjorgemota/gurudamatricula/robot/format"
9           "github.com/fjorgemota/gurudamatricula/util/coder"
10  )
11
```

```
12 //go:generate codecgen -o $GOPACKAGE.generated.go campus.go course.go
   ↪ discipline_offer.go discipline.go habilitation.go period.go room.go
   ↪ teacher.go team.go
13
14 type Callback func(data Entity) error
15 type convertFromTeam func(universityID string, team *format.Team, save
   ↪ Callback) error
16 type convertFromHabilitation func(universityID string, habilitation
   ↪ *format.Habilitation, save Callback) error
17 type Getter func() Entity
18
19 const chunkSize = 1000
20
21 var (
22         errConflictingInformation = errors.New("ddiffer: Conflicting
          ↪ information")
23         errDuplicateInformation   = errors.New("ddiffer: Duplicate
          ↪ information")
24         errDifferentType          = errors.New("ddiffer: Different types
          ↪ cannot be compared")
25         errInvalidKind            = errors.New("ddiffer: Only structs can
          ↪ be compared")
26         errInvalidSource          = errors.New("ddiffer: Source invalid")
27         errInvalidType            = errors.New("ddiffer: Type invalid to
          ↪ merge")
28         errInvalidID              = errors.New("ddiffer: ID invalid to
          ↪ merge")
29         errGetterNotFound         = errors.New("ddiffer: Getter not
          ↪ found")
30         errConversorNotFound      = errors.New("ddiffer: Conversor not
          ↪ found")
31 )
32
33 type Limit struct {
34         Items int
35         Size  int
36 }
37
38 type Entity interface {
```

```
39          GetID() string
40          GetPeriodID() string
41          GetUniversityID() string
42          Merge(Entity) error
43          EncodeTo(coder.StreamingEncoder) error
44          DecodeFrom(decoder coder.StreamingDecoder) error
45          Equal(Entity) bool
46          Clean()
47          Put()
48  }
49
50  type Handler interface {
51          GetStreamingDecoder(io.Reader) coder.StreamingDecoder
52          GetStreamingEncoder(io.WriteCloser) coder.StreamingEncoder
53  }
54
55  func init() {
56          gob.Register(DiffCampus{})
57          gob.Register(DiffCourse{})
58          gob.Register(DiffDiscipline{})
59          gob.Register(DiffDisciplineOffer{})
60          gob.Register(DiffHabilitation{})
61          gob.Register(DiffPeriod{})
62          gob.Register(DiffRoom{})
63          gob.Register(DiffTeacher{})
64          gob.Register(DiffTeam{})
65  }
```

Arquivo campus.go

```
1   package ddiffer
2
3   import (
4           "reflect"
5           "sort"
6           "sync"
7
8           "github.com/fjorgemota/gurudamatricula/models/keygen"
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/util/coder"
```

```
)

type DiffCampus struct {
        ID           string           `json:"id"`
        UniversityID string           `json:"university_id"`
        Period       DiffForeignKey   `json:"period"`
        Campus       format.Campus    `json:"campus"`
        Disciplines  []DiffForeignKey `json:"discipline_ids"`
}

var diffCampusPool = sync.Pool{
        New: func() interface{} {
                return &DiffCampus{
                        Disciplines: make([]DiffForeignKey, 0, 50),
                }
        },
}

func (dc *DiffCampus) GetID() string {
        return dc.ID
}

func (dc *DiffCampus) GetUniversityID() string {
        return dc.UniversityID
}

func (dc *DiffCampus) GetPeriodID() string {
        return dc.Period.ID
}

func (dc *DiffCampus) Equal(data Entity) bool {
        campus, ok := data.(*DiffCampus)
        return ok && reflect.DeepEqual(campus, dc)
}

func (dc *DiffCampus) Merge(data Entity) error {
        campus, ok := data.(*DiffCampus)
        err := errInvalidType
        if ok && campus != nil {
```

```
50                  err = errInvalidID
51                  if campus.ID == dc.ID {
52                          err = newConflictError(TargetCampus, dc)
53                          if dc.Campus == campus.Campus {
54                                  dc.Disciplines = append(dc.Disciplines,
                                    ↪  campus.Disciplines...)
55                                  err = nil
56                          }
57
58                  }
59          }
60          return err
61  }
62
63  func (dc *DiffCampus) EncodeTo(encoder coder.StreamingEncoder) error {
64          sort.Sort(foreignKeySorter(dc.Disciplines))
65          return encoder.Encode(dc)
66  }
67
68  func (dc *DiffCampus) DecodeFrom(decoder coder.StreamingDecoder) error {
69          dc.Clean()
70          return decoder.Decode(dc)
71  }
72
73  func (dc *DiffCampus) Clean() {
74          *dc = DiffCampus{
75                  Disciplines: cleanForeignKeys(dc.Disciplines, 2000),
76          }
77  }
78  func (dc *DiffCampus) Put() {
79          dc.Clean()
80          diffCampusPool.Put(dc)
81  }
82
83  func toDiffCampus(universityID string, team *format.Team, save Callback)
    ↪  error {
84          id, err := keygen.GenerateCampusID(universityID, team.Period,
            ↪  team.Campus)
85          var periodID, disciplineOfferID string
```

```
86          if err == nil {
87                  periodID, err = keygen.GeneratePeriodID(universityID,
                    ↪  team.Period)
88          }
89          if err == nil {
90                  disciplineOfferID, err =
                    ↪  keygen.GenerateDisciplineOfferID(universityID,
                    ↪  team.Period, team.Campus, format.DisciplineOffer{ID:
                    ↪  team.Discipline.ID})
91          }
92          if err == nil {
93                  result := diffCampusPool.Get().(*DiffCampus)
94                  result.ID = id
95                  result.UniversityID = universityID
96                  result.Campus = team.Campus
97                  result.Period = DiffForeignKey{
98                          ID:   periodID,
99                          Name: team.Period.Name,
100                 }
101                 result.Disciplines = append(result.Disciplines,
                    ↪  DiffForeignKey{
102                         ID:   disciplineOfferID,
103                         Name: team.Discipline.Name,
104                 })
105                 err = save(result)
106         }
107         return err
108 }
109
110 func GetDiffCampus() Entity {
111         return diffCampusPool.Get().(*DiffCampus)
112 }
```

Arquivo changed.go

```
1  package ddiffer
2
3  import (
4          "reflect"
5  )
```

```
6
7  func Changed(x, y interface{}) ([]string, error) {
8          xv := reflect.ValueOf(x)
9          yv := reflect.ValueOf(y)
10         for xv.Kind() == reflect.Ptr {
11                 xv = xv.Elem()
12         }
13         for yv.Kind() == reflect.Ptr {
14                 yv = yv.Elem()
15         }
16         var err error
17         var fields []string
18         if err == nil && (xv.Kind() != reflect.Struct || yv.Kind() !=
           ↪  reflect.Struct) {
19                 err = newErrInvalidKind()
20         }
21         if err == nil && xv.Type() != yv.Type() {
22                 err = newErrDifferentType()
23         }
24         if err == nil {
25                 xt := xv.Type()
26                 nfields := xt.NumField()
27                 for i := 0; i < nfields; i++ {
28                         xf := xv.Field(i)
29                         yf := yv.Field(i)
30                         if !reflect.DeepEqual(xf.Interface(),
                           ↪  yf.Interface()) {
31                                 fields = append(fields, xt.Field(i).Name)
32                         }
33                 }
34         }
35         return fields, err
36 }
```

Arquivo comparer.go

```
1  package ddiffer
2
3  import (
4          "fmt"
```

```
5           "io"
6
7           "github.com/fjorgemota/gurudamatricula/util/coder"
8           "github.com/fjorgemota/gurudamatricula/util/file"
9           "github.com/pkg/errors"
10  )
11
12  type ComparerOptions struct {
13          Limits           Limit
14          Target           TargetType
15          OldSource        string
16          NewSource        string
17          CreatedTemplate  string
18          ModifiedTemplate string
19          DeletedTemplate  string
20          ArchivedPeriods  []string
21  }
22
23  type ComparerResult struct {
24          Target    TargetType
25          OldSource string
26          NewSource string
27          Created   []string
28          Modified  []string
29          Deleted   []string
30  }
31
32  func IsComparerResultEmpty(cr ComparerResult) bool {
33          return len(cr.Created) == 0 && len(cr.Modified) == 0 &&
            ↪  len(cr.Deleted) == 0
34  }
35
36  type compareState struct {
37          oldEntity, newEntity Entity
38          oldSource, newSource coder.StreamingDecoder
39          oldEnd, newEnd       bool
40          oldErr, newErr       error
41          archivedPeriods      map[string]struct{}
42  }
```

```
43
44 func (state *compareState) loadOld() error {
45 loadOld:
46         state.oldErr = state.oldEntity.DecodeFrom(state.oldSource)
47         if state.oldErr != nil {
48                 state.oldEntity.Put()
49                 state.oldEntity = nil
50         } else if _, ok :=
         ↪  state.archivedPeriods[state.oldEntity.GetPeriodID()]; ok {
51                 goto loadOld
52         }
53         if errors.Cause(state.oldErr) == io.EOF {
54                 state.oldEnd = true
55                 state.oldErr = nil
56         }
57         return state.oldErr
58 }
59
60 func (state *compareState) loadNew() error {
61 loadNew:
62         state.newErr = state.newEntity.DecodeFrom(state.newSource)
63         if state.newErr != nil {
64                 state.newEntity.Put()
65                 state.newEntity = nil
66         } else if _, ok :=
         ↪  state.archivedPeriods[state.newEntity.GetPeriodID()]; ok {
67                 goto loadNew
68         }
69         if errors.Cause(state.newErr) == io.EOF {
70                 state.newEnd = true
71                 state.newErr = nil
72         }
73         return state.newErr
74 }
75
76 func (state *compareState) loadBoth() error {
77         err := state.loadOld()
78         if err == nil {
79                 err = state.loadNew()
```

```
80          }
81          return err
82  }
83
84  func (state *compareState) shouldIgnore() bool {
85          return state.sameEntity() &&
86                  state.oldEntity.Equal(state.newEntity)
87  }
88
89  func (state *compareState) sameEntity() bool {
90          return state.oldEntity != nil && state.newEntity != nil &&
91                  state.oldEntity.GetPeriodID() ==
                    ↪  state.newEntity.GetPeriodID() &&
92                  state.oldEntity.GetID() == state.newEntity.GetID()
93  }
94
95  func (state *compareState) isDeleted() bool {
96          return state.newEntity == nil ||
97                  (state.oldEntity != nil &&
98                          (state.oldEntity.GetPeriodID() <
                            ↪  state.newEntity.GetPeriodID() ||
99                                  state.oldEntity.GetID() <
                                    ↪  state.newEntity.GetID()))
100 }
101
102 func (state *compareState) isCreated() bool {
103         return state.oldEntity == nil ||
104                 (state.newEntity != nil &&
105                         (state.oldEntity.GetPeriodID() >
                           ↪  state.newEntity.GetPeriodID() ||
106                                 state.oldEntity.GetID() >
                                   ↪  state.newEntity.GetID()))
107 }
108
109 func checkSizeLimit(encoder coder.StreamingEncoder, counter
    ↪  *counterWriter, options ComparerOptions) error {
110         var err error
111         if counter != nil && options.Limits.Size > 0 && counter.counter >
            ↪  options.Limits.Size {
```

```
112                 err = encoder.Close()
113         }
114         return err
115 }
116
117 func Compare(manager file.Manager, handler Handler, options
    ↪  ComparerOptions) (ComparerResult, error) {
118         archivedPeriods := make(map[string]struct{},
            ↪  len(options.ArchivedPeriods))
119         for _, period := range options.ArchivedPeriods {
120                 archivedPeriods[period] = struct{}{}
121         }
122         getter, err := GetGetter(options.Target)
123         var oldSource, newSource coder.StreamingDecoder
124         var oldReader, newReader io.ReadCloser
125         if options.OldSource != "" {
126                 if err == nil {
127                         oldReader, err =
                            ↪  manager.Reader(options.OldSource)
128                 }
129                 if err == nil {
130                         oldSource =
                            ↪  handler.GetStreamingDecoder(oldReader)
131                 }
132         }
133         if options.NewSource != "" {
134                 if err == nil {
135                         newReader, err =
                            ↪  manager.Reader(options.NewSource)
136                 }
137                 if err == nil {
138                         newSource =
                            ↪  handler.GetStreamingDecoder(newReader)
139                 }
140         }
141         state := &compareState{
142                 oldEnd:          oldSource == nil,
143                 newEnd:          newSource == nil,
144                 oldSource:       oldSource,
```

```
145                 newSource:       newSource,
146                 archivedPeriods: archivedPeriods,
147         }
148         if err == nil && !state.oldEnd {
149                 state.oldEntity = getter()
150                 err = state.loadOld()
151         }
152         if err == nil && !state.newEnd {
153                 state.newEntity = getter()
154                 err = state.loadNew()
155         }
156         var result ComparerResult
157         result.Target = options.Target
158         result.OldSource = options.OldSource
159         result.NewSource = options.NewSource
160         var counterCreated, counterModified, counterDeleted
            ↪  *counterWriter
161         var partCreated, partModified, partDeleted int
162         var created, modified, deleted coder.StreamingEncoder
163         if options.CreatedTemplate != "" {
164                 created = coder.NewSplitterEncoder(func()
                    ↪  (coder.StreamingEncoder, error) {
165                         tmpName := fmt.Sprintf(options.CreatedTemplate,
                            ↪  partCreated)
166                         partCreated++
167                         result.Created = append(result.Created, tmpName)
168                         writer, localErr := manager.Writer(tmpName)
169                         if localErr == nil {
170                                 counterCreated = &counterWriter{writer:
                                    ↪  writer, counter: 0}
171                         }
172                         var partEncoder coder.StreamingEncoder
173                         if localErr == nil {
174                                 partEncoder =
                                    ↪  handler.GetStreamingEncoder(counterCreated)
175                         }
176                         return partEncoder, localErr
177                 }, options.Limits.Items)
178         }
```

```
179         if options.ModifiedTemplate != "" {
180                 modified = created
181                 if options.CreatedTemplate != options.ModifiedTemplate {
182                         modified = coder.NewSplitterEncoder(func()
                        ↪  (coder.StreamingEncoder, error) {
183                                 tmpName :=
                                ↪  fmt.Sprintf(options.ModifiedTemplate,
                                ↪  partModified)
184                                 partModified++
185                                 result.Modified = append(result.Modified,
                                ↪  tmpName)
186                                 writer, localErr :=
                                ↪  manager.Writer(tmpName)
187                                 if localErr == nil {
188                                         counterModified =
                                        ↪  &counterWriter{writer: writer,
                                        ↪  counter: 0}
189                                 }
190                                 var partEncoder coder.StreamingEncoder
191                                 if localErr == nil {
192                                         partEncoder =
                                        ↪  handler.GetStreamingEncoder(counterModified)
193                                 }
194                                 return partEncoder, localErr
195                         }, options.Limits.Items)
196                 }
197         }
198         if options.DeletedTemplate != "" {
199                 deleted = coder.NewSplitterEncoder(func()
                ↪  (coder.StreamingEncoder, error) {
200                         tmpName := fmt.Sprintf(options.DeletedTemplate,
                        ↪  partDeleted)
201                         partDeleted++
202                         result.Deleted = append(result.Deleted, tmpName)
203                         writer, localErr := manager.Writer(tmpName)
204                         if localErr == nil {
205                                 counterDeleted = &counterWriter{writer:
                                ↪  writer, counter: 0}
206                         }
```

```
207                         var partEncoder coder.StreamingEncoder
208                         if localErr == nil {
209                                 partEncoder =
                                    ↪ handler.GetStreamingEncoder(counterDeleted)
210                         }
211                         return partEncoder, localErr
212                 }, options.Limits.Items)
213         }
214         for err == nil && (!state.oldEnd || !state.newEnd) {
215                 if state.shouldIgnore() {
216                         err = state.loadBoth()
217                 } else if state.sameEntity() {
218                         if modified != nil {
219                                 err = state.newEntity.EncodeTo(modified)
220                                 if err == nil {
221                                         err = checkSizeLimit(modified,
                                            ↪ counterModified, options)
222                                 }
223                         }
224                         if err == nil {
225                                 err = state.loadBoth()
226                         }
227                 } else if state.isDeleted() {
228                         if deleted != nil {
229                                 err = state.oldEntity.EncodeTo(deleted)
230                                 if err == nil {
231                                         err = checkSizeLimit(deleted,
                                            ↪ counterDeleted, options)
232                                 }
233                         }
234                         if err == nil {
235                                 err = state.loadOld()
236                         }
237                 } else if state.isCreated() {
238                         if created != nil {
239                                 err = state.newEntity.EncodeTo(created)
240                                 if err == nil {
241                                         err = checkSizeLimit(created,
                                            ↪ counterCreated, options)
```

```
242                                 }
243                         }
244                         if err == nil {
245                                 err = state.loadNew()
246                         }
247                 }
248         }
249         for _, encoder := range []coder.StreamingEncoder{created,
          ↪  modified, deleted} {
250                 if err == nil && encoder != nil {
251                         err = encoder.Flush()
252                 }
253         }
254         for _, closer := range []io.Closer{oldReader, newReader, created,
          ↪  modified, deleted} {
255                 if err == nil && closer != nil {
256                         err = closer.Close()
257                 }
258         }
259         return result, err
260 }
```

Arquivo counter.go

```
1  package ddiffer
2
3  import "io"
4
5  type counterWriter struct {
6          writer  io.WriteCloser
7          counter int
8  }
9
10 func (cw *counterWriter) Close() error {
11         cw.counter = 0
12         return cw.writer.Close()
13 }
14
15 func (cw *counterWriter) Write(bs []byte) (int, error) {
16         cw.counter += len(bs)
```

```
17          return cw.writer.Write(bs)
18  }
```

Arquivo course.go

```
1   package ddiffer
2
3   import (
4           "reflect"
5           "sort"
6           "sync"
7
8           "github.com/fjorgemota/gurudamatricula/models/keygen"
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/util/coder"
11  )
12
13  type DiffCourse struct {
14          ID            string           `json:"id"`
15          UniversityID  string           `json:"university_id"`
16          Course        format.Course    `json:"course"`
17          Habilitations []DiffForeignKey `json:"courses_ids"`
18  }
19
20  var diffCoursePool = sync.Pool{
21          New: func() interface{} {
22                  return new(DiffCourse)
23          },
24  }
25
26  func (dc *DiffCourse) GetID() string {
27          return dc.ID
28  }
29
30  func (dc *DiffCourse) GetUniversityID() string {
31          return dc.UniversityID
32  }
33
34  func (dc *DiffCourse) GetPeriodID() string {
35          return ""
```

```
36 }
37
38 func (dc *DiffCourse) Equal(data Entity) bool {
39         course, ok := data.(*DiffCourse)
40         return ok && reflect.DeepEqual(course, dc)
41 }
42
43 func (dc *DiffCourse) Merge(data Entity) error {
44         course, ok := data.(*DiffCourse)
45         err := errInvalidType
46         if ok && course != nil {
47                 err = errInvalidID
48                 if course.ID == dc.ID {
49                         dc.Habilitations = append(dc.Habilitations,
                        ↪  course.Habilitations...)
50                         err = nil
51                 }
52         }
53         return err
54 }
55
56 func (dc *DiffCourse) EncodeTo(encoder coder.StreamingEncoder) error {
57         sort.Sort(foreignKeySorter(dc.Habilitations))
58         return encoder.Encode(dc)
59 }
60
61 func (dc *DiffCourse) DecodeFrom(decoder coder.StreamingDecoder) error {
62         dc.Clean()
63         return decoder.Decode(dc)
64 }
65
66 func (dc *DiffCourse) Clean() {
67         *dc = DiffCourse{
68                 Habilitations: cleanForeignKeys(dc.Habilitations, 8),
69         }
70 }
71
72 func (dc *DiffCourse) Put() {
73         dc.Clean()
```

```
74          diffCoursePool.Put(dc)
75  }
76
77  func toDiffCourse(universityID string, habilitation *format.Habilitation,
    ↪  save Callback) error {
78          id, err := keygen.GenerateCourseID(universityID,
            ↪  habilitation.Course)
79          var habID string
80          if err == nil {
81                  habID, err = keygen.GenerateHabilitationID(universityID,
                    ↪  *habilitation)
82          }
83          if err == nil {
84                  result := diffCoursePool.Get().(*DiffCourse)
85                  result.ID = id
86                  result.UniversityID = universityID
87                  result.Course = habilitation.Course
88                  result.Habilitations = append(result.Habilitations,
                    ↪  DiffForeignKey{
89                          ID:   habID,
90                          Name: habilitation.Name,
91                  })
92                  err = save(result)
93          }
94          return err
95  }
96
97  func GetDiffCourse() Entity {
98          return diffCoursePool.Get().(*DiffCourse)
99  }
```

Arquivo discipline.go

```
1   package ddiffer
2
3   import (
4           "reflect"
5           "sync"
6
7           "github.com/fjorgemota/gurudamatricula/robot/format/pool"
```

```
8
9           "github.com/fjorgemota/gurudamatricula/models/keygen"
10          "github.com/fjorgemota/gurudamatricula/robot/format"
11          "github.com/fjorgemota/gurudamatricula/util/coder"
12  )
13
14  type DiffDiscipline struct {
15          ID           string            `json:"id"`
16          GenericID    string            `json:"generic_id"`
17          UniversityID string            `json:"university_id"`
18          Habilitation DiffForeignKey    `json:"habilitation"`
19          Course       DiffForeignKey    `json:"course"`
20          Step         DiffForeignKey    `json:"step"`
21          Discipline   format.Discipline `json:"discipline"`
22  }
23
24  var diffDisciplinePool = sync.Pool{
25          New: func() interface{} {
26                  return new(DiffDiscipline)
27          },
28  }
29
30  func (dd *DiffDiscipline) GetID() string {
31          return dd.ID
32  }
33
34  func (dd *DiffDiscipline) GetUniversityID() string {
35          return dd.UniversityID
36  }
37
38  func (dd *DiffDiscipline) GetPeriodID() string {
39          return ""
40  }
41
42  func (dd *DiffDiscipline) Equal(data Entity) bool {
43          discipline, ok := data.(*DiffDiscipline)
44          return ok && reflect.DeepEqual(dd, discipline)
45  }
46
```

```
47 func (dd *DiffDiscipline) Merge(data Entity) error {
48         return newDuplicateError(TargetDiscipline, dd)
49 }
50
51 func (dd *DiffDiscipline) EncodeTo(encoder coder.StreamingEncoder) error
   ↪ {
52         return encoder.Encode(dd)
53 }
54
55 func (dd *DiffDiscipline) DecodeFrom(decoder coder.StreamingDecoder)
   ↪  error {
56         dd.Clean()
57         return decoder.Decode(dd)
58 }
59
60 func (dd *DiffDiscipline) Clean() {
61         pool.CleanDiscipline(&dd.Discipline)
62         *dd = DiffDiscipline{
63                 Discipline: dd.Discipline,
64         }
65 }
66
67 func (dd *DiffDiscipline) Put() {
68         dd.Clean()
69         diffDisciplinePool.Put(dd)
70 }
71
72 func toDiffDiscipline(universityID string, habilitation
   ↪  *format.Habilitation, save Callback) error {
73         var err error
74         habilitationRef := DiffForeignKey{
75                 Name: habilitation.Name,
76         }
77         courseRef := DiffForeignKey{
78                 Name: habilitation.Course.Name,
79         }
80         habilitationRef.ID, err =
          ↪  keygen.GenerateHabilitationID(universityID, *habilitation)
81         if err == nil {
```

```
82                courseRef.ID, err = keygen.GenerateCourseID(universityID,
                  ↪  habilitation.Course)
83        }
84        for _, step := range habilitation.Steps {
85                var stepID string
86                if err == nil {
87                        stepID, err = keygen.GenerateStepID(universityID,
                          ↪  *habilitation, step)
88                }
89                stepRef := DiffForeignKey{
90                        Name: step.Name,
91                }
92                if err == nil {
93                        stepRef.ID = stepID
94                }
95                for _, discipline := range step.Disciplines {
96                        var id, genericID string
97                        if err == nil {
98                                id, err =
                                  ↪  keygen.GenerateDisciplineID(universityID,
                                  ↪  discipline)
99                        }
100                       if err == nil {
101                               genericID, err =
                                  ↪  keygen.GenerateDisciplineGenericID(universityID,
                                  ↪  discipline.GenericID)
102                       }
103                       for i, table := range discipline.Tables {
104                               var tableID string
105                               if err == nil {
106                                       tableID, err =
                                          ↪  keygen.GenerateTableID(universityID,
                                          ↪  id, table)
107                               }
108                               if err == nil {
109                                       discipline.Tables[i].ID = tableID
110                               }
111                       }
112                       for i, related := range discipline.Related {
```

```
113                                 for j, relatedItem := range related.Items
                                    ↪ {
114                                         value := relatedItem.Value
115                                         if err == nil {
116                                                 switch relatedItem.Type {
117                                                 case "discipline_id":
118                                                         value, err =
                                                            ↪ keygen.GenerateDiscipli
                                                            ↪ format.Discipline{ID:
                                                            ↪ value})
119                                                 case
                                                    ↪ "discipline_generic_id":
120                                                         value, err =
                                                            ↪ keygen.GenerateDiscipli
                                                            ↪ value)
121                                                 }
122                                         }
123                                         if err == nil {
124                                                 discipline.Related[i].Items[j].Valu
                                                    ↪ = value
125                                         }
126                                 }
127                         }
128                         if err == nil {
129                                 result :=
                                    ↪ diffDisciplinePool.Get().(*DiffDiscipline)
130                                 result.ID = id
131                                 result.GenericID = genericID
132                                 result.UniversityID = universityID
133                                 result.Habilitation = habilitationRef
134                                 result.Step = stepRef
135                                 result.Course = courseRef
136                                 pool.CopyDiscipline(&result.Discipline,
                                    ↪ &discipline)
137                                 err = save(result)
138                         }
139                 }
140         }
141         return err
```

```
142 }
143
144 func GetDiffDiscipline() Entity {
145         return diffDisciplinePool.Get().(*DiffDiscipline)
146 }
```

Arquivo discipline_offer.go

```
1  package ddiffer
2
3  import (
4          "reflect"
5          "sync"
6
7          "github.com/fjorgemota/gurudamatricula/models/keygen"
8          "github.com/fjorgemota/gurudamatricula/robot/format"
9          "github.com/fjorgemota/gurudamatricula/robot/format/pool"
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11 )
12
13 type DiffDisciplineOffer struct {
14         ID              string                  `json:"id"`
15         GenericID       string                  `json:"generic_id"`
16         UniversityID    string                  `json:"university_id"`
17         Period          DiffForeignKey          `json:"period"`
18         Campus          DiffForeignKey          `json:"campus"`
19         DisciplineOffer format.DisciplineOffer `json:"discipline_offer"`
20         Teams           []format.Team           `json:"teams"`
21         GenericIDs      []DiffGenericID         `json:"generic_ids"`
22 }
23
24 var diffDisciplineOfferPool = sync.Pool{
25         New: func() interface{} {
26                 return &DiffDisciplineOffer{}
27         },
28 }
29
30 func (ddo *DiffDisciplineOffer) GetID() string {
31         return ddo.ID
32 }
```

```
33
34 func (ddo *DiffDisciplineOffer) GetUniversityID() string {
35         return ddo.UniversityID
36 }
37
38 func (ddo *DiffDisciplineOffer) GetPeriodID() string {
39         return ddo.Period.ID
40 }
41
42 func (ddo *DiffDisciplineOffer) GetGenericID(id string) string {
43         return getGenericID(ddo.GenericIDs, id)
44 }
45
46 func (ddo *DiffDisciplineOffer) Equal(data Entity) bool {
47         disciplineOffer, ok := data.(*DiffDisciplineOffer)
48         return ok && reflect.DeepEqual(disciplineOffer, ddo)
49 }
50
51 func (ddo *DiffDisciplineOffer) Merge(data Entity) error {
52         disciplineOffer, ok := data.(*DiffDisciplineOffer)
53         err := errInvalidType
54         if ok && disciplineOffer != nil {
55                 err = errInvalidID
56                 if disciplineOffer.ID == ddo.ID {
57                         start := len(ddo.Teams)
58                         ddo.GenericIDs =
                        ↪  prepareGenericIDs(append(ddo.GenericIDs,
                        ↪  disciplineOffer.GenericIDs...))
59                         ddo.Teams = append(ddo.Teams, make([]format.Team,
                        ↪  len(disciplineOffer.Teams))...)
60                         for i, team := range disciplineOffer.Teams {
61                                 pool.CopyTeam(&ddo.Teams[i+start], &team)
62                         }
63                         err = nil
64                 }
65         }
66         return err
67 }
68
```

```
69 func (ddo *DiffDisciplineOffer) EncodeTo(encoder coder.StreamingEncoder)
   ↪  error {
70         return encoder.Encode(ddo)
71 }
72
73 func (ddo *DiffDisciplineOffer) DecodeFrom(decoder
   ↪  coder.StreamingDecoder) error {
74         ddo.Clean()
75         return decoder.Decode(ddo)
76 }
77
78 func (ddo *DiffDisciplineOffer) Clean() {
79         teams := ddo.Teams
80         for i := range teams {
81                 pool.CleanTeam(&teams[i])
82         }
83         if teams == nil {
84                 teams = make([]format.Team, 0)
85         }
86         *ddo = DiffDisciplineOffer{Teams: teams[:0]}
87 }
88
89 func (ddo *DiffDisciplineOffer) Put() {
90         ddo.Clean()
91         diffDisciplineOfferPool.Put(ddo)
92 }
93
94 func toDiffDisciplineOffer(universityID string, team *format.Team, save
   ↪  Callback) error {
95         id, err := keygen.GenerateDisciplineOfferID(universityID,
           ↪  team.Period, team.Campus, team.Discipline)
96         var periodID, campusID, genericID string
97         if err == nil {
98                 periodID, err = keygen.GeneratePeriodID(universityID,
                   ↪  team.Period)
99         }
100        if err == nil {
101                campusID, err = keygen.GenerateCampusID(universityID,
                   ↪  team.Period, team.Campus)
```

```
102         }
103         if err == nil {
104                 genericID, err =
                    ↪ keygen.GenerateDisciplineGenericID(universityID,
                    ↪ team.Discipline.GenericID)
105         }
106         if err == nil {
107                 result :=
                    ↪ diffDisciplineOfferPool.Get().(*DiffDisciplineOffer)
108                 result.ID = id
109                 result.GenericID = genericID
110                 result.UniversityID = universityID
111                 result.Period = DiffForeignKey{
112                         ID:   periodID,
113                         Name: team.Period.Name,
114                 }
115                 result.Campus = DiffForeignKey{
116                         ID:   campusID,
117                         Name: team.Campus.Name,
118                 }
119                 result.DisciplineOffer = format.DisciplineOffer{
120                         ID:        id,
121                         GenericID: genericID,
122                         Code:      team.Discipline.Code,
123                         Name:      team.Discipline.Name,
124                 }
125                 if cap(result.Teams) == 0 {
126                         result.Teams = append(result.Teams,
                            ↪ format.Team{})
127                 } else {
128                         result.Teams = result.Teams[:1]
129                 }
130                 pool.CopyTeam(&result.Teams[0], team)
131                 if err == nil {
132                         result.Teams[0].ID, err =
                            ↪ keygen.GenerateTeamID(universityID,
                            ↪ result.Teams[0])
133                 }
134                 if err == nil && len(team.Course.ID) > 0 {
```

```
135                     result.Teams[0].Course.ID, err =
                        ↪  keygen.GenerateCourseID(universityID,
                        ↪  format.Course{ID: team.Course.ID})
136             }
137             for i, table := range team.Tables {
138                     var tableID string
139                     if err == nil {
140                             tableID, err =
                                ↪  keygen.GenerateTableID(universityID,
                                ↪  id, table)
141                     }
142                     if err == nil {
143                             result.Teams[0].Tables[i].ID = tableID
144                     }
145             }
146             for i, schedule := range team.Schedules {
147                     if len(schedule.Room.ID) == 0 {
148                             continue
149                     }
150                     var roomPeriodID, roomID string
151                     if err == nil {
152                             roomPeriodID, err =
                                ↪  keygen.GenerateRoomByPeriodID(universityID,
                                ↪  team.Period, team.Campus,
                                ↪  schedule.Room)
153                     }
154                     if err == nil {
155                             roomID, err =
                                ↪  keygen.GenerateRoomID(universityID,
                                ↪  team.Campus, schedule.Room)
156                     }
157                     if err == nil {
158                             result.Teams[0].Schedules[i].Room.ID =
                                ↪  roomID
159                             result.GenericIDs =
                                ↪  append(result.GenericIDs,
                                ↪  DiffGenericID{
160                                     ID:        roomID,
161                                     GenericID: roomPeriodID,
```

```
162                                     })
163                             }
164                     }
165                     for i, teacher := range team.Teachers {
166                             var teacherPeriodID, teacherID string
167                             if err == nil {
168                                     teacherPeriodID, err =
                                        ↪ keygen.GenerateTeacherByPeriodID(universityID,
                                        ↪ team.Period, teacher)
169                             }
170                             if err == nil {
171                                     teacherID, err =
                                        ↪ keygen.GenerateTeacherID(universityID,
                                        ↪ teacher)
172                             }
173                             if err == nil {
174                                     result.Teams[0].Teachers[i].ID =
                                        ↪ teacherID
175                                     result.GenericIDs =
                                        ↪ append(result.GenericIDs,
                                        ↪ DiffGenericID{
176                                             ID:        teacherID,
177                                             GenericID: teacherPeriodID,
178                                     })
179                             }
180                     }
181                     result.GenericIDs = prepareGenericIDs(result.GenericIDs)
182                     if err == nil {
183                             err = save(result)
184                     }
185             }
186             return err
187     }
188
189     func GetDiffDisciplineOffer() Entity {
190             return diffDisciplineOfferPool.Get().(*DiffDisciplineOffer)
191     }
```

Arquivo errors.go

```
1  package ddiffer
2
3  import (
4          "fmt"
5
6          "github.com/pkg/errors"
7  )
8
9  type entityError struct {
10         dataset TargetType
11         id      string
12         err     error
13 }
14
15 func (k entityError) Error() string {
16         return fmt.Sprintf("error on entity %s and dataset '%s': %s",
           ↪  k.id, k.dataset.String(), k.err.Error())
17 }
18
19 func (k entityError) Cause() error {
20         return k.err
21 }
22
23 func newConflictError(dataset TargetType, entity Entity) error {
24         return entityError{dataset, entity.GetID(),
           ↪  errConflictingInformation}
25 }
26
27 func IsErrConflictingInformation(err error) bool {
28         return errors.Cause(err) == errConflictingInformation
29 }
30
31 func newDuplicateError(dataset TargetType, entity Entity) error {
32         return entityError{dataset, entity.GetID(),
           ↪  errDuplicateInformation}
33 }
34
35 func IsErrDuplicateInformation(err error) bool {
36         return errors.Cause(err) == errDuplicateInformation
```

```
}

func newErrDifferentType() error {
        return errDifferentType
}

func newErrInvalidKind() error {
        return errInvalidKind
}

func IsErrInvalidKind(err error) bool {
        return errors.Cause(err) == errInvalidKind
}

func IsErrDifferentType(err error) bool {
        return errors.Cause(err) == errDifferentType
}
```

Arquivo foreign_key.go

```
package ddiffer

import "sort"

type DiffForeignKey struct {
        ID   string `json:"id"`
        Name string `json:"name"`
}

func cleanForeignKeys(result []DiffForeignKey, limit int)
↪  []DiffForeignKey {
        if result == nil {
                result = make([]DiffForeignKey, 0, limit)
        }
        for i := range result {
                result[i] = DiffForeignKey{}
        }
        return result[:0]
}

```

```
20 type foreignKeySorter []DiffForeignKey
21
22 func (fks foreignKeySorter) Len() int {
23         return len(fks)
24 }
25
26 func (fks foreignKeySorter) Less(i, j int) bool {
27         return fks[i].ID < fks[i].ID
28 }
29
30 func (fks foreignKeySorter) Swap(i, j int) {
31         fks[i], fks[j] = fks[j], fks[i]
32 }
33
34 var _ sort.Interface = foreignKeySorter{}
```

Arquivo generic_id.go

```
1  package ddiffer
2
3  import "sort"
4
5  type DiffGenericID struct {
6          ID        string `json:"id"`
7          GenericID string `json:"generic_id"`
8  }
9
10 type genericIDSorter []DiffGenericID
11
12 func (gks genericIDSorter) Len() int {
13         return len(gks)
14 }
15
16 func (gks genericIDSorter) Less(i, j int) bool {
17         return gks[i].ID < gks[i].ID
18 }
19
20 func (gks genericIDSorter) Swap(i, j int) {
21         gks[i], gks[j] = gks[j], gks[i]
22 }
```

```
23
24 func prepareGenericIDs(items []DiffGenericID) []DiffGenericID {
25         sort.Sort(genericIDSorter(items))
26         var lastID string
27         result := items[:0]
28         for _, item := range items {
29                 if item.ID != lastID {
30                         lastID = item.ID
31                         result = append(result, item)
32                 }
33         }
34         return result
35 }
36
37 func getGenericID(items []DiffGenericID, id string) string {
38         for _, item := range items {
39                 if item.ID == id {
40                         return item.GenericID
41                 }
42         }
43         return ""
44 }
```

Arquivo habilitation.go

```
1  package ddiffer
2
3  import (
4          "reflect"
5          "sync"
6
7          "github.com/fjorgemota/gurudamatricula/models/keygen"
8          "github.com/fjorgemota/gurudamatricula/robot/format"
9          "github.com/fjorgemota/gurudamatricula/robot/format/pool"
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11 )
12
13 type DiffHabilitation struct {
14         ID           string              `json:"id"`
15         UniversityID string              `json:"university_id"`
```

```
16          Course       DiffForeignKey      `json:"course"`
17          Habilitation format.Habilitation `json:"habilitation"`
18  }
19
20  var diffHabilitationPool = sync.Pool{
21          New: func() interface{} {
22                  return new(DiffHabilitation)
23          },
24  }
25
26  func (dh *DiffHabilitation) GetID() string {
27          return dh.ID
28  }
29
30  func (dh *DiffHabilitation) GetUniversityID() string {
31          return dh.UniversityID
32  }
33
34  func (dh *DiffHabilitation) GetPeriodID() string {
35          return ""
36  }
37
38  func (dh *DiffHabilitation) Equal(data Entity) bool {
39          habilitation, ok := data.(*DiffHabilitation)
40          return ok && reflect.DeepEqual(habilitation, dh)
41  }
42
43  func (dh *DiffHabilitation) Merge(data Entity) error {
44          return newDuplicateError(TargetHabilitation, dh)
45  }
46
47  func (dh *DiffHabilitation) EncodeTo(encoder coder.StreamingEncoder)
    ↪  error {
48          return encoder.Encode(dh)
49  }
50
51  func (dh *DiffHabilitation) DecodeFrom(decoder coder.StreamingDecoder)
    ↪  error {
52          dh.Clean()
```

```
53          return decoder.Decode(dh)
54  }
55
56  func (dh *DiffHabilitation) Clean() {
57          pool.CleanHabilitation(&dh.Habilitation)
58          *dh = DiffHabilitation{
59                  Habilitation: dh.Habilitation,
60          }
61  }
62  func (dh *DiffHabilitation) Put() {
63          dh.Clean()
64          diffHabilitationPool.Put(dh)
65  }
66
67  func toDiffHabilitation(universityID string, habilitation
    ↪  *format.Habilitation, save Callback) error {
68          id, err := keygen.GenerateHabilitationID(universityID,
            ↪  *habilitation)
69          var courseID string
70          if err == nil {
71                  courseID, err = keygen.GenerateCourseID(universityID,
                    ↪  habilitation.Course)
72          }
73          for i, table := range habilitation.Tables {
74                  var tableID string
75                  if err == nil {
76                          tableID, err =
                            ↪  keygen.GenerateTableID(universityID, id,
                            ↪  table)
77                  }
78                  if err == nil {
79                          habilitation.Tables[i].ID = tableID
80                  }
81          }
82          for i, step := range habilitation.Steps {
83                  var stepID string
84                  if err == nil {
85                          stepID, err = keygen.GenerateStepID(universityID,
                            ↪  *habilitation, step)
```

```
86                  }
87                  if err == nil {
88                          habilitation.Steps[i].ID = stepID
89                  }
90                  for j, discipline := range step.Disciplines {
91                          var disciplineID string
92                          if err == nil {
93                                  disciplineID, err =
                                    ↪  keygen.GenerateDisciplineID(universityID,
                                    ↪  discipline)
94                          }
95                          if err == nil {
96                                  habilitation.Steps[i].Disciplines[j] =
                                    ↪  format.Discipline{
97                                          ID:   disciplineID,
98                                          Name: discipline.Name,
99                                  }
100                         }
101                 }
102         }
103         if err == nil {
104                 result := diffHabilitationPool.Get().(*DiffHabilitation)
105                 result.ID = id
106                 result.Course = DiffForeignKey{
107                         ID:   courseID,
108                         Name: habilitation.Course.Name,
109                 }
110                 result.UniversityID = universityID
111                 pool.CopyHabilitation(&result.Habilitation, habilitation)
112                 err = save(result)
113         }
114         return err
115 }
116
117 func GetDiffHabilitation() Entity {
118         return diffHabilitationPool.Get().(*DiffHabilitation)
119 }
```

Arquivo init.go

```
package ddiffer

import (
        "encoding/gob"
)

func init() {
        // Registers SourceType and TargetType
        gob.Register(SourceHabilitations)
        gob.Register(TargetCampus)
        gob.Register([]TargetType{})
        gob.Register([]PipelineConverterResult{})
        gob.Register(PipelineConverterOptions{})
        gob.Register(SplitterOptions{})
        gob.Register(ConverterOptions{})
        gob.Register(Limit{})
        gob.Register(ComparerOptions{})
        gob.Register(ComparerResult{})
}
```

Arquivo loaders.go

```
package ddiffer

//go:generate stringer -type=SourceType -output=loaders_source_string.go
↪  -trimprefix=Source
//go:generate stringer -type=TargetType -output=loaders_target_string.go
↪  -trimprefix=Target

type SourceType int8

const (
        _                       = iota // ignore first value by assigning
        ↪  to blank identifier
        SourceOffers SourceType = 1 << iota
        SourceHabilitations
)

type TargetType int16

```

```
16 const (
17         _                       = iota // ignore first value by assigning
           ↪  to blank identifier
18         TargetCampus TargetType = 1 << iota
19         TargetDisciplineOffer
20         TargetPeriod
21         TargetRoom
22         TargetTeacher
23         TargetTeam
24         TargetCourse
25         TargetDiscipline
26         TargetHabilitation
27 )
28
29 var teamConversors = map[TargetType]convertFromTeam{
30         TargetCampus:          toDiffCampus,
31         TargetDisciplineOffer: toDiffDisciplineOffer,
32         TargetPeriod:          toDiffPeriod,
33         TargetRoom:            toDiffRoom,
34         TargetTeacher:         toDiffTeacher,
35         TargetTeam:            toDiffTeam,
36 }
37
38 var getters = map[TargetType]Getter{
39         TargetCampus:          GetDiffCampus,
40         TargetDisciplineOffer: GetDiffDisciplineOffer,
41         TargetPeriod:          GetDiffPeriod,
42         TargetRoom:            GetDiffRoom,
43         TargetTeacher:         GetDiffTeacher,
44         TargetTeam:            GetDiffTeam,
45         TargetCourse:          GetDiffCourse,
46         TargetDiscipline:      GetDiffDiscipline,
47         TargetHabilitation:    GetDiffHabilitation,
48 }
49
50 var habilitationConversors = map[TargetType]convertFromHabilitation{
51         TargetCourse:       toDiffCourse,
52         TargetDiscipline:   toDiffDiscipline,
53         TargetHabilitation: toDiffHabilitation,
```

```
54 }
55
56 func GetGetter(target TargetType) (Getter, error) {
57         var result Getter
58         err := errGetterNotFound
59         if getter, ok := getters[target]; ok {
60                 result = getter
61                 err = nil
62         }
63         return result, err
64 }
65
66 func Get(target TargetType) (Entity, error) {
67         var result Entity
68         getter, err := GetGetter(target)
69         if err == nil {
70                 result = getter()
71         }
72         return result, err
73 }
74
75 func getTeamConversor(target TargetType) (convertFromTeam, error) {
76         var conversor convertFromTeam
77         err := errConversorNotFound
78         if converter, ok := teamConversors[target]; ok {
79                 conversor = converter
80                 err = nil
81         }
82         return conversor, err
83 }
84
85 func getHabilitationConversor(target TargetType) (convertFromHabilitation,
   ↪  error) {
86         var conversor convertFromHabilitation
87         err := errConversorNotFound
88         if converter, ok := habilitationConversors[target]; ok {
89                 conversor = converter
90                 err = nil
91         }
```

```
92          return conversor, err
93  }
```

Arquivo loaders_source_string.go

```
1   // Code generated by "stringer -type=SourceType
    ↪  -output=loaders_source_string.go -trimprefix=Source"; DO NOT EDIT.
2
3   package ddiffer
4
5   import "strconv"
6
7   const (
8           _SourceType_name_0 = "Offers"
9           _SourceType_name_1 = "Habilitations"
10  )
11
12  func (i SourceType) String() string {
13          switch {
14          case i == 2:
15                  return _SourceType_name_0
16          case i == 4:
17                  return _SourceType_name_1
18          default:
19                  return "SourceType(" + strconv.FormatInt(int64(i), 10) +
                    ↪  ")"
20          }
21  }
```

Arquivo loaders_target_string.go

```
1   // Code generated by "stringer -type=TargetType
    ↪  -output=loaders_target_string.go -trimprefix=Target"; DO NOT EDIT.
2
3   package ddiffer
4
5   import "strconv"
6
7   const (
```

```
        _TargetType_name_0 = "Campus"
        _TargetType_name_1 = "DisciplineOffer"
        _TargetType_name_2 = "Period"
        _TargetType_name_3 = "Room"
        _TargetType_name_4 = "Teacher"
        _TargetType_name_5 = "Team"
        _TargetType_name_6 = "Course"
        _TargetType_name_7 = "Discipline"
        _TargetType_name_8 = "Habilitation"
)

func (i TargetType) String() string {
        switch {
        case i == 2:
                return _TargetType_name_0
        case i == 4:
                return _TargetType_name_1
        case i == 8:
                return _TargetType_name_2
        case i == 16:
                return _TargetType_name_3
        case i == 32:
                return _TargetType_name_4
        case i == 64:
                return _TargetType_name_5
        case i == 128:
                return _TargetType_name_6
        case i == 256:
                return _TargetType_name_7
        case i == 512:
                return _TargetType_name_8
        default:
                return "TargetType(" + strconv.FormatInt(int64(i), 10) +
                ↪  ")"
        }
}
```

Arquivo merge_sorted.go

```
package ddiffer
```

```
2
3  import (
4          "container/heap"
5          "io"
6
7          "github.com/fjorgemota/gurudamatricula/util/coder"
8          "github.com/pkg/errors"
9  )
10
11 func MergeSorted(decoders []coder.StreamingDecoder, encoder
   ↪  coder.StreamingEncoder, get Getter) error {
12         var err error
13         entities := make([]mergeSorterEntity, len(decoders))
14         handler := &mergeSorter{collection: entities}
15         for _, decoder := range decoders {
16                 if err == nil {
17                         data := get()
18                         err = data.DecodeFrom(decoder)
19                         if err == nil {
20                                 heap.Push(handler, mergeSorterEntity{
21                                         entity:  data,
22                                         decoder: decoder,
23                                 })
24                         } else if errors.Cause(err) == io.EOF {
25                                 err = nil
26                         }
27                 }
28         }
29         var lastEntity Entity
30         for !handler.Empty() && err == nil {
31                 var reuse bool
32                 actual := handler.collection[0]
33                 if lastEntity != nil && actual.entity.GetPeriodID() ==
                   ↪  lastEntity.GetPeriodID() && actual.entity.GetID() ==
                   ↪  lastEntity.GetID() {
34                         err = lastEntity.Merge(actual.entity)
35                         reuse = true
36                 } else if lastEntity != nil {
37                         err = lastEntity.EncodeTo(encoder)
```

```
38                      lastEntity.Put()
39                      lastEntity = actual.entity
40              } else {
41                      lastEntity = actual.entity
42              }
43
44              if err == nil {
45                      // NOTE: Este bloco de codigo deveria reduzir
                        ↪  chamadas ao Pool, o que
46                      // a principio é uma boa ideia, MAS causa um
                        ↪  problema complicado de resolver:
47                      // O código acima só roda porque lastEntity é
                        ↪  projetado como uma CÚPIA da
48                      // entidade, mas, se a gente modifica
                        ↪  actual.entity aqui (e nós estavamos
49                      //       modificando), então lastEntity TAMBÉM é
                        ↪  modificado, o que significa que
50                      // de repente lastEntity = actual.entity é sempre
                        ↪  válido E portanto seus
51                      // IDs são iguais também, causando problemas no
                        ↪  mínimo bizarros e bastante
52                      // similares a uma condição de corrida, MAS sem
                        ↪  envolver condições de corrida
53                      // e portanto sendo indetectável ao race detector
                        ↪  do Golang.
54                      if reuse {
55                              err =
                                ↪  actual.entity.DecodeFrom(actual.decoder)
56                              if err == nil {
57                                      heap.Fix(handler, 0)
58                              } else if errors.Cause(err) == io.EOF {
59                                      actual.entity.Put()
60                                      heap.Pop(handler)
61                                      err = nil
62                              }
63                      } else {
64                              data := get()
65                              err = data.DecodeFrom(actual.decoder)
66                              if err == nil {
```

```
67                                          handler.collection[0].entity =
                                            ↪  data
68                                          heap.Fix(handler, 0)
69                                  } else if errors.Cause(err) == io.EOF {
70                                          heap.Pop(handler)
71                                          err = nil
72                                  }
73                          }
74                  }
75          }
76          if err == nil && lastEntity != nil {
77                  err = lastEntity.EncodeTo(encoder)
78          }
79          return err
80  }
```

Arquivo period.go

```
1   package ddiffer
2
3   import (
4           "reflect"
5           "sync"
6
7           "github.com/fjorgemota/gurudamatricula/models/keygen"
8           "github.com/fjorgemota/gurudamatricula/robot/format"
9           "github.com/fjorgemota/gurudamatricula/util/coder"
10  )
11
12  type DiffPeriod struct {
13          ID           string          `json:"id"`
14          UniversityID string          `json:"university_id"`
15          Period       format.Period   `json:"period"`
16          Campi        []format.Campus `json:"campi"`
17          campi        map[string]struct{}
18  }
19
20  var diffPeriodPool = sync.Pool{
21          New: func() interface{} {
22                  return new(DiffPeriod)
```

```
23          },
24  }
25
26  func (dp *DiffPeriod) GetID() string {
27          return dp.ID
28  }
29
30  func (dp *DiffPeriod) GetUniversityID() string {
31          return dp.UniversityID
32  }
33
34  func (dp *DiffPeriod) GetPeriodID() string {
35          return dp.ID
36  }
37
38  func (dp *DiffPeriod) Equal(data Entity) bool {
39          period, ok := data.(*DiffPeriod)
40          return ok && reflect.DeepEqual(period, dp)
41  }
42
43  func (dp *DiffPeriod) Merge(data Entity) error {
44          period, ok := data.(*DiffPeriod)
45          err := errInvalidType
46          if ok && period != nil {
47                  if dp.Period.Name != period.Period.Name {
48                          err = newConflictError(TargetPeriod, dp)
49                  } else {
50                          err = nil
51                          var zero struct{}
52                          if dp.campi == nil {
53                                  dp.campi = make(map[string]struct{},
                                    ↪  len(dp.Campi))
54                                  for _, campus := range dp.Campi {
55                                          dp.campi[campus.ID] = zero
56                                  }
57                          }
58                          for _, campus := range period.Campi {
59                                  if _, ok := dp.campi[campus.ID]; !ok {
```

```
60                                              dp.Campi = append(dp.Campi,
                                                ↪  campus)
61                                              dp.campi[campus.ID] = zero
62                                      }
63                              }
64                      }
65              }
66              return err
67      }
68
69      func (dp *DiffPeriod) EncodeTo(encoder coder.StreamingEncoder) error {
70              return encoder.Encode(dp)
71      }
72
73      func (dp *DiffPeriod) DecodeFrom(decoder coder.StreamingDecoder) error {
74              dp.Clean()
75              return decoder.Decode(dp)
76      }
77
78      func (dp *DiffPeriod) Clean() {
79              campiList := dp.Campi
80              campiMap := dp.campi
81              if cap(campiList) > 100 {
82                      campiList = make([]format.Campus, 0, 100)
83                      campiMap = make(map[string]struct{}, 100)
84              }
85              for i := range campiList {
86                      delete(campiMap, campiList[i].ID)
87                      campiList[i] = format.Campus{}
88              }
89              *dp = DiffPeriod{
90                      campi: campiMap,
91                      Campi: campiList[:0],
92              }
93      }
94
95      func (dp *DiffPeriod) Put() {
96              dp.Clean()
97              diffPeriodPool.Put(dp)
```

```
}

func toDiffPeriod(universityID string, team *format.Team, save Callback)
↪  error {
        id, err := keygen.GeneratePeriodID(universityID, team.Period)
        var campusID string
        if err == nil {
                campusID, err = keygen.GenerateCampusID(universityID,
                ↪  team.Period, team.Campus)
        }
        if err == nil {
                result := diffPeriodPool.Get().(*DiffPeriod)
                result.ID = id
                result.UniversityID = universityID
                result.Period = team.Period
                result.Campi = []format.Campus{team.Campus}
                result.Campi[0].ID = campusID
                err = save(result)
        }
        return err
}

func GetDiffPeriod() Entity {
        return diffPeriodPool.Get().(*DiffPeriod)
}
```

Arquivo pipeline.go

```
package ddiffer

import (
        "context"
        "fmt"
        "io"
        "strconv"
        "strings"

        "github.com/fjorgemota/gurudamatricula/util/coder"
        "github.com/fjorgemota/gurudamatricula/util/file"
        "github.com/fjorgemota/gurudamatricula/util/pipeline"
```

```
13 )
14
15 type PipelineDependencies interface {
16         Handler
17         GetManager(context.Context) (file.Manager, error)
18 }
19
20 type PipelineConverterResult struct {
21         UniversityID string
22         Target       TargetType
23         Result       string
24 }
25
26 type PipelineConverterOptions struct {
27         UniversityID   string
28         TargetTemplate string
29         Limits         Limit
30         Name           string
31         Source         SourceType
32         Targets        []TargetType
33 }
34
35 type PipelineGlobalOptions struct {
36         ConvertersLimits     Limit
37         ComparersLimits      Limit
38         MergeSourcesLimit    int
39         SplitConvertTaskName string
40         ConvertTaskName      string
41         SortTaskName         string
42         MergeTaskName        string
43         ResultTaskName       string
44         CompareTaskName      string
45 }
46
47 type pipelineSplitter struct {
48         dependency     Handler
49         initial, final bool
50 }
51
```

```
func (pw pipelineSplitter) GetStreamingDecoder(reader io.Reader)
↪  coder.StreamingDecoder {
        if pw.initial {
                return coder.NewJSONDecoder(reader)
        }
        return coder.NewGOBStreamingDecoder(reader)
}
func (pw pipelineSplitter) GetStreamingEncoder(writer io.WriteCloser)
↪  coder.StreamingEncoder {
        if pw.final {
                return pw.dependency.GetStreamingEncoder(writer)
        }
        return coder.NewGOBStreamingEncoder(writer)
}

type pipelineHandler struct {
        handler PipelineDependencies
        options PipelineGlobalOptions
}

func getDefaultLimit(limits ...Limit) Limit {
        var result Limit
        for _, limit := range limits {
                if result.Items == 0 {
                        result.Items = limit.Items
                }
                if result.Size == 0 {
                        result.Size = limit.Size
                }
        }
        return result
}

func getDefaultPipelineGlobalOptions(opts PipelineGlobalOptions)
↪  PipelineGlobalOptions {
        if opts.MergeSourcesLimit == 0 {
                opts.MergeSourcesLimit = 5
        }
        if opts.SplitConvertTaskName == "" {
```

```
88                  opts.SplitConvertTaskName = "ddiffer.split_convert"
89          }
90          if opts.ConvertTaskName == "" {
91                  opts.ConvertTaskName = "ddiffer.convert"
92          }
93          if opts.SortTaskName == "" {
94                  opts.SortTaskName = "ddiffer.sort"
95          }
96          if opts.MergeTaskName == "" {
97                  opts.MergeTaskName = "ddiffer.merge"
98          }
99          if opts.ResultTaskName == "" {
100                 opts.ResultTaskName = "ddiffer.result"
101         }
102         if opts.CompareTaskName == "" {
103                 opts.CompareTaskName = "ddiffer.compare"
104         }
105         return opts
106 }
107
108 func (ph pipelineHandler) result(ctx context.Context, _ pipeline.Pipeline,
    ↪  universityID string, allTargets, targets []TargetType, partFiles
    ↪  []string, results ...string) ([]PipelineConverterResult, error) {
109         manager, err := ph.handler.GetManager(ctx)
110         if err == nil {
111                 err = file.DeleteFiles(manager, partFiles)
112         }
113         var result []PipelineConverterResult
114         if err == nil {
115                 targetResults := make(map[TargetType]string,
                  ↪  len(targets))
116                 for i, target := range targets {
117                         targetResults[target] = results[i]
118                 }
119                 for _, target := range allTargets {
120                         result = append(result, PipelineConverterResult{
121                                 UniversityID: universityID,
122                                 Target:       target,
123                                 Result:       targetResults[target],
```

```
124                         })
125                 }
126         }
127         return result, err
128 }
129
130 func (ph pipelineHandler) splitConvert(ctx context.Context, pipe
    ↪  pipeline.Pipeline, options PipelineConverterOptions)
    ↪  (pipeline.FutureID, error) {
131         manager, err := ph.handler.GetManager(ctx)
132         partString := strings.ToLower(options.Source.String())
133         options.Limits = getDefaultLimit(options.Limits,
            ↪  ph.options.ConvertersLimits)
134         var filenames []string
135         if err == nil {
136                 splitOptions := SplitterOptions{
137                         Name:           options.Name,
138                         Limits:         options.Limits,
139                         TargetTemplate: options.Name + "." + partString +
                        ↪  "-split-part-%d",
140                 }
141                 handler := pipelineSplitter{
142                         dependency: ph.handler,
143                         initial:    true,
144                 }
145                 filenames, err = SplitSource(manager, handler,
                    ↪  options.Source, splitOptions)
146         }
147         var resultFilenames []pipeline.FutureID
148         var selectedTargets, targets []TargetType
149         if err == nil {
150                 switch options.Source {
151                 case SourceOffers:
152                         for _, target := range options.Targets {
153                                 if _, ok := teamConversors[target]; ok {
154                                         targets = append(targets, target)
155                                 }
156                         }
157                 case SourceHabilitations:
```

```
158                     for _, target := range options.Targets {
159                             if _, ok :=
                                ↪  habilitationConversors[target]; ok {
160                                     targets = append(targets, target)
161                             }
162                     }
163             }
164             if len(targets) == 0 {
165                     switch options.Source {
166                     case SourceOffers:
167                             for target := range teamConversors {
168                                     targets = append(targets, target)
169                             }
170                     case SourceHabilitations:
171                             for target := range
                                ↪  habilitationConversors {
172                                     targets = append(targets, target)
173                             }
174                     }
175             }
176             final := len(filenames) == 1
177             for _, target := range targets {
178                     if err == nil {
179                             convertOptions := ConverterOptions{
180                                     UniversityID:
                                        ↪  options.UniversityID,
181                                     Target:       target,
182                             }
183                             targetString :=
                                ↪  strings.ToLower(target.String())
184                             var results []pipeline.FutureID
185                             for _, filename := range filenames {
186                                     var result []pipeline.FutureID
187                                     if err == nil {
188                                             targetFile := filename +
                                                ↪  "-" + targetString
189                                             if final {
```

```
190                                             targetFile =
                                                ↪ fmt.Sprintf(options.Tar
                                                ↪ targetString)
191                                     }
192                                     splitOptions :=
                                        ↪ SplitterOptions{
193                                             Name:
                                                ↪ filename,
194                                             Limits:
                                                ↪ options.Limits,
195                                             TargetTemplate:
                                                ↪ filename +
                                                ↪ "." +
                                                ↪ targetString
                                                ↪ +
                                                ↪ "-convert-part-%d",
196                                     }
197                                     result, err =
                                        ↪ pipe.Dispatch(ph.options.Conver
                                        ↪ splitOptions,
                                        ↪ convertOptions,
                                        ↪ targetFile,
                                        ↪ options.Source,
                                        ↪ final)
198                             }
199                             if err == nil {
200                                     if final {
201                                             resultFilenames =
                                                ↪ append(resultFilenames,
                                                ↪ result[0])
202                                             selectedTargets =
                                                ↪ append(selectedTargets,
                                                ↪ target)
203                                     } else {
204                                             results =
                                                ↪ append(results,
                                                ↪ result[0])
205                                     }
206                             }
```

```
207                                 }
208                                 if len(filenames) > 1 {
209                                         var result []pipeline.FutureID
210                                         if err == nil {
211                                                 args :=
                                                    ↪ make([]interface{},
                                                    ↪ len(results)+3)
212                                                 args[0] = true
213                                                 args[1] = target
214                                                 args[2] =
                                                    ↪ fmt.Sprintf(options.TargetTemplate,
                                                    ↪ targetString)
215                                                 for i, arg := range
                                                    ↪ results {
216                                                         args[i+3] = arg
217                                                 }
218                                                 result, err =
                                                    ↪ pipe.Dispatch(ph.options.MergeTaskNa
                                                    ↪ args...)
219                                         }
220                                         if err == nil {
221                                                 resultFilenames =
                                                    ↪ append(resultFilenames,
                                                    ↪ result[0])
222                                                 selectedTargets =
                                                    ↪ append(selectedTargets,
                                                    ↪ target)
223                                         }
224                                 }
225                         }
226                 }
227         }
228         var finalResult pipeline.FutureID
229         if err == nil {
230                 var finalResults []pipeline.FutureID
231                 args := make([]interface{}, len(resultFilenames)+4)
232                 args[0] = options.UniversityID
233                 args[1] = targets
234                 args[2] = selectedTargets
```

```
235                 args[3] = filenames
236                 for i, arg := range resultFilenames {
237                         args[i+4] = arg
238                 }
239                 finalResults, err =
                    ↪  pipe.Dispatch(ph.options.ResultTaskName, args...)
240                 if err == nil {
241                         finalResult = finalResults[0]
242                 }
243         }
244         return finalResult, err
245 }
246
247 func (ph pipelineHandler) convert(ctx context.Context, pipe
    ↪  pipeline.Pipeline, splitOptions SplitterOptions, convertOptions
    ↪  ConverterOptions, target string, source SourceType, final bool)
    ↪  (string, error) {
248         manager, err := ph.handler.GetManager(ctx)
249         handler := pipelineSplitter{
250                 dependency: ph.handler,
251         }
252         var filenames []string
253         if err == nil {
254                 filenames, err = Convert(manager, handler, source,
                    ↪  splitOptions, convertOptions)
255         }
256         if len(filenames) == 0 && err == nil {
257                 // In case we don't have any files created, just create a
                    ↪  empty file as target
258                 var writer io.WriteCloser
259                 var encoder coder.StreamingEncoder
260                 // If we're finalizing processing, we need to mark it to
                    ↪  get the correct
261                 // encoder..
262                 handler.final = final
263                 writer, err = manager.Writer(target)
264                 if err == nil {
265                         encoder = handler.GetStreamingEncoder(writer)
266                 }
```

```
267                 if err == nil {
268                         err = encoder.Flush()
269                 }
270                 if err == nil {
271                         err = encoder.Close()
272                 }
273         }
274         var results []pipeline.FutureID
275         finalSort := final && len(filenames) == 1
276         for _, filename := range filenames {
277                 var result []pipeline.FutureID
278                 if err == nil {
279                         sortedFilename := filename + "-sorted"
280                         if len(filenames) == 1 {
281                                 // If we have only one filename, sort the
                                    ↪  file in the target
282                                 // specified
283                                 sortedFilename = target
284                         }
285                         result, err =
                            ↪  pipe.Dispatch(ph.options.SortTaskName,
                            ↪  finalSort, convertOptions.Target, filename,
                            ↪  sortedFilename)
286                 }
287                 if err == nil && len(filenames) > 1 {
288                         results = append(results, result[0])
289                 }
290         }
291         if err == nil && len(filenames) > 1 {
292                 args := make([]interface{}, len(results)+3)
293                 args[0] = final
294                 args[1] = convertOptions.Target
295                 args[2] = target
296                 for i, arg := range results {
297                         args[i+3] = arg
298                 }
299                 if err == nil {
300                         _, err = pipe.Dispatch(ph.options.MergeTaskName,
                            ↪  args...)
```

```
301                 }
302         }
303         return target, err
304 }
305
306 func (ph pipelineHandler) sort(ctx context.Context, _ pipeline.Pipeline,
    ↪  final bool, target TargetType, fileSource, fileTarget string) (string,
    ↪  error) {
307         manager, err := ph.handler.GetManager(ctx)
308         handler := pipelineSplitter{
309                 dependency: ph.handler,
310                 final:      final,
311         }
312         var reader file.ReaderSeekerCloser
313         var writer io.WriteCloser
314         var decoder coder.StreamingDecoder
315         var encoder coder.StreamingEncoder
316         if err == nil {
317                 reader, err = manager.Reader(fileSource)
318         }
319         if err == nil {
320                 writer, err = manager.Writer(fileTarget)
321         }
322         if err == nil {
323                 decoder = handler.GetStreamingDecoder(reader)
324                 encoder = handler.GetStreamingEncoder(writer)
325         }
326         if err == nil {
327                 err = SortMerge(decoder, encoder, target)
328         }
329         if err == nil {
330                 err = encoder.Flush()
331         }
332         if err == nil {
333                 err = encoder.Close()
334         }
335         if err == nil {
336                 err = reader.Close()
337         }
```

```
338         if err == nil {
339                 err = manager.Delete(fileSource)
340         }
341         return fileTarget, err
342 }
343
344 func (ph pipelineHandler) merge(ctx context.Context, pipe
    ↪  pipeline.Pipeline, final bool, target TargetType, fileTarget string,
    ↪  fileSources ...string) (string, error) {
345         var err error
346         if ph.options.MergeSourcesLimit > 1 && len(fileSources) >
            ↪  ph.options.MergeSourcesLimit {
347                 var oldQueue, newQueue []interface{}
348                 oldQueue = make([]interface{}, len(fileSources))
349                 for i, source := range fileSources {
350                         oldQueue[i] = source
351                 }
352                 count := 0
353                 for len(oldQueue) > ph.options.MergeSourcesLimit {
354                         var queue [][]interface{}
355                         for i := 0; i < len(oldQueue); i +=
                            ↪  ph.options.MergeSourcesLimit {
356                                 if err == nil {
357                                         max := i +
                                            ↪  ph.options.MergeSourcesLimit
358                                         if max > len(oldQueue) {
359                                                 newQueue =
                                                    ↪  append(newQueue,
                                                    ↪  oldQueue[i:]...)
360                                                 continue
361                                         }
362                                         queue = append(queue,
                                            ↪  oldQueue[i:max])
363                                 }
364                         }
365                         for _, items := range queue {
366                                 args := make([]interface{}, 3,
                                    ↪  len(items)+3)
367                                 args[0] = false
```

```
368                                 args[1] = target
369                                 count++
370                                 args[2] = fileTarget + "-merge-part-" +
                                    ↪  strconv.Itoa(count)
371                                 args = append(args, items...)
372                                 var result []pipeline.FutureID
373                                 if err == nil {
374                                         result, err =
                                            ↪  pipe.Dispatch(ph.options.MergeTaskName,
                                            ↪  args...)
375                                 }
376                                 if err == nil {
377                                         newQueue = append(newQueue,
                                            ↪  result[0])
378                                 }
379                         }
380                         for i := range oldQueue {
381                                 oldQueue[i] = nil
382                         }
383                         oldQueue, newQueue = newQueue, oldQueue[:0]
384                 }
385                 finalArgs := make([]interface{}, 3, 3+len(oldQueue))
386                 finalArgs[0] = final
387                 finalArgs[1] = target
388                 finalArgs[2] = fileTarget
389                 finalArgs = append(finalArgs, oldQueue...)
390                 if err == nil {
391                         _, err = pipe.Dispatch(ph.options.MergeTaskName,
                            ↪  finalArgs...)
392                 }
393                 return fileTarget, err
394         }
395         var manager file.Manager
396         manager, err = ph.handler.GetManager(ctx)
397         var writer io.WriteCloser
398         var readers []io.ReadCloser
399         var decoders []coder.StreamingDecoder
400         var encoder coder.StreamingEncoder
401         var getter Getter
```

```
402         handler := pipelineSplitter{
403                 dependency: ph.handler,
404                 final:      final,
405         }
406         if err == nil {
407                 getter, err = GetGetter(target)
408         }
409         for _, fileSource := range fileSources {
410                 var reader io.ReadCloser
411                 if err == nil {
412                         reader, err = manager.Reader(fileSource)
413                 }
414                 if err == nil {
415                         decoder := handler.GetStreamingDecoder(reader)
416                         decoders = append(decoders, decoder)
417                         readers = append(readers, reader)
418                 }
419         }
420         if err == nil {
421                 writer, err = manager.Writer(fileTarget)
422         }
423         if err == nil {
424                 encoder = handler.GetStreamingEncoder(writer)
425                 err = MergeSorted(decoders, encoder, getter)
426         }
427         if err == nil {
428                 err = encoder.Flush()
429         }
430         if err == nil {
431                 err = encoder.Close()
432         }
433         for _, reader := range readers {
434                 if err == nil {
435                         err = reader.Close()
436                 }
437         }
438         if err == nil {
439                 err = file.DeleteFiles(manager, fileSources)
440         }
```

```
441         return fileTarget, err
442 }
443
444 func (ph pipelineHandler) compare(ctx context.Context, pipe
    ↪  pipeline.Pipeline, options ComparerOptions) (ComparerResult, error) {
445         manager, err := ph.handler.GetManager(ctx)
446         var result ComparerResult
447         if err == nil {
448                 options.Limits = getDefaultLimit(options.Limits,
                    ↪  ph.options.ComparersLimits)
449                 result, err = Compare(manager, ph.handler, options)
450         }
451         return result, err
452 }
453
454 func (ph pipelineHandler) Convert(pipe pipeline.Pipeline, options
    ↪  PipelineConverterOptions) (pipeline.FutureID, error) {
455         var result pipeline.FutureID
456         results, err := pipe.Dispatch(ph.options.SplitConvertTaskName,
            ↪  options)
457         if err == nil {
458                 result = results[0]
459         }
460         return result, err
461 }
462
463 func (ph pipelineHandler) Compare(pipe pipeline.Pipeline, options
    ↪  ComparerOptions) (pipeline.FutureID, error) {
464         var result pipeline.FutureID
465         results, err := pipe.Dispatch(ph.options.CompareTaskName,
            ↪  options)
466         if err == nil {
467                 result = results[0]
468         }
469         return result, err
470 }
471
472 type PipelineHandler interface {
```

```
473         Convert(pipe pipeline.Pipeline, options PipelineConverterOptions)
            ↪  (pipeline.FutureID, error)
474         Compare(pipe pipeline.Pipeline, options ComparerOptions)
            ↪  (pipeline.FutureID, error)
475 }
476
477 func NewPipelineHandler(disp pipeline.Dispatcher, handler
    ↪  PipelineDependencies, opts PipelineGlobalOptions) (PipelineHandler,
    ↪  error) {
478         opts = getDefaultPipelineGlobalOptions(opts)
479         result := pipelineHandler{
480                 handler: handler,
481                 options: opts,
482         }
483         err := disp.Register(opts.SplitConvertTaskName,
            ↪  result.splitConvert)
484         if err == nil {
485                 err = disp.Register(opts.ConvertTaskName, result.convert)
486         }
487         if err == nil {
488                 err = disp.Register(opts.SortTaskName, result.sort)
489         }
490         if err == nil {
491                 err = disp.Register(opts.MergeTaskName, result.merge)
492         }
493         if err == nil {
494                 err = disp.Register(opts.ResultTaskName, result.result)
495         }
496         if err == nil {
497                 err = disp.Register(opts.CompareTaskName, result.compare)
498         }
499         return result, err
500 }
```

Arquivo room.go

```
1 package ddiffer
2
3 import (
4         "fmt"
```

```
5           "reflect"
6           "sync"
7
8           "github.com/fjorgemota/gurudamatricula/models/keygen"
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/util/coder"
11  )
12
13  type DiffRoom struct {
14          ID           string         `json:"id"`
15          GenericID    string         `json:"generic_id"`
16          UniversityID string         `json:"university_id"`
17          Period       DiffForeignKey `json:"period_id"`
18          Campus       DiffForeignKey `json:"campus"`
19          Room         format.Room    `json:"room"`
20  }
21
22  var diffRoomPool = sync.Pool{
23          New: func() interface{} {
24                  return new(DiffRoom)
25          },
26  }
27
28  func (dr *DiffRoom) GetID() string {
29          return dr.ID
30  }
31
32  func (dr *DiffRoom) GetUniversityID() string {
33          return dr.UniversityID
34  }
35
36  func (dr *DiffRoom) GetPeriodID() string {
37          return dr.Period.ID
38  }
39
40  func (dr *DiffRoom) Equal(data Entity) bool {
41          room, ok := data.(*DiffRoom)
42          result := ok && reflect.DeepEqual(room, dr)
43          if !result {
```

```
                fmt.Println(dr, " != ", data)
                panic("hmmm")
        }
        return result
}

func (dr *DiffRoom) Merge(data Entity) error {
        return nil
}

func (dr *DiffRoom) EncodeTo(encoder coder.StreamingEncoder) error {
        return encoder.Encode(dr)
}

func (dr *DiffRoom) DecodeFrom(decoder coder.StreamingDecoder) error {
        dr.Clean()
        return decoder.Decode(dr)
}

func (dr *DiffRoom) Clean() {
        *dr = DiffRoom{}
}

func (dr *DiffRoom) Put() {
        dr.Clean()
        diffRoomPool.Put(dr)
}

func toDiffRoom(universityID string, team *format.Team, save Callback)
↪  error {
        periodID, err := keygen.GeneratePeriodID(universityID,
        ↪  team.Period)
        var campusID string
        if err == nil {
                campusID, err = keygen.GenerateCampusID(universityID,
                ↪  team.Period, team.Campus)
        }
        periodRef := DiffForeignKey{
                ID:   periodID,
```

```
80                  Name: team.Period.Name,
81          }
82          campusRef := DiffForeignKey{
83                  ID:   campusID,
84                  Name: team.Campus.Name,
85          }
86          for _, schedule := range team.Schedules {
87                  if schedule.Room.ID == "" {
88                          continue
89                  }
90                  var id, genericID string
91                  if err == nil {
92                          id, err =
                            ↪ keygen.GenerateRoomByPeriodID(universityID,
                            ↪ team.Period, team.Campus, schedule.Room)
93                  }
94                  if err == nil {
95                          genericID, err =
                            ↪ keygen.GenerateRoomID(universityID,
                            ↪ team.Campus, schedule.Room)
96                  }
97                  if err == nil {
98                          result := diffRoomPool.Get().(*DiffRoom)
99                          result.ID = id
100                         result.GenericID = genericID
101                         result.UniversityID = universityID
102                         result.Period = periodRef
103                         result.Campus = campusRef
104                         result.Room = schedule.Room
105                         err = save(result)
106                 }
107         }
108         return err
109 }
110
111 func GetDiffRoom() Entity {
112         return diffRoomPool.Get().(*DiffRoom)
113 }
```

Arquivo sort_merge.go

```
1  package ddiffer
2
3  import (
4          "io"
5          "sort"
6
7          "github.com/fjorgemota/gurudamatricula/util/coder"
8          "github.com/pkg/errors"
9  )
10
11 func SortMerge(decoder coder.StreamingDecoder, encoder
   ↪  coder.StreamingEncoder, target TargetType) error {
12         var handler sorter
13         var err error
14         for err == nil {
15                 var entity Entity
16                 entity, err = Get(target)
17                 if err == nil {
18                         err = entity.DecodeFrom(decoder)
19                 }
20                 if err == nil {
21                         handler = append(handler, entity)
22                 }
23         }
24         if errors.Cause(err) == io.EOF {
25                 err = nil
26         }
27         if len(handler) == 0 {
28                 return err
29         }
30         if err == nil {
31                 sort.Sort(handler)
32         }
33         lastEntity := handler[0]
34         i := 1
35         for err == nil && i < handler.Len() {
36                 actual := handler[i]
```

```
37                  if lastEntity != nil && actual.GetID() ==
                    ↪  lastEntity.GetID() {
38                          err = lastEntity.Merge(actual)
39                  } else {
40                          err = lastEntity.EncodeTo(encoder)
41                          lastEntity = actual
42                  }
43                  i++
44          }
45          if err == nil && lastEntity != nil {
46                  err = lastEntity.EncodeTo(encoder)
47          }
48          for _, item := range handler {
49                  item.Put()
50          }
51          return errors.WithMessage(errors.WithStack(err), "Error when
            ↪  sorting and merging "+target.String())
52  }
```

Arquivo sorter.go

```
1   package ddiffer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/util/coder"
5   )
6
7   type mergeSorterEntity struct {
8           entity  Entity
9           decoder coder.StreamingDecoder
10  }
11
12  type mergeSorter struct {
13          collection []mergeSorterEntity
14          length     int
15  }
16
17  func (s *mergeSorter) Swap(i, j int) {
18          s.collection[i], s.collection[j] = s.collection[j],
            ↪  s.collection[i]
```

```
19 }
20
21 func (s *mergeSorter) Less(i, j int) bool {
22         entityI := s.collection[i].entity
23         entityJ := s.collection[j].entity
24         if entityI.GetPeriodID() != entityJ.GetPeriodID() {
25                 return entityI.GetPeriodID() < entityJ.GetPeriodID()
26         }
27         return entityI.GetID() < entityJ.GetID()
28 }
29
30 func (s *mergeSorter) Len() int {
31         return s.length
32 }
33
34 func (s *mergeSorter) Push(data interface{}) {
35         s.collection[s.length] = data.(mergeSorterEntity)
36         s.length++
37 }
38
39 func (s *mergeSorter) Pop() interface{} {
40         s.length--
41         return s.collection[s.length]
42 }
43
44 func (s *mergeSorter) Empty() bool {
45         return s.length == 0
46 }
47
48 type sorter []Entity
49
50 func (s sorter) Swap(i, j int) {
51         s[i], s[j] = s[j], s[i]
52 }
53
54 func (s sorter) Less(i, j int) bool {
55         sI := s[i]
56         sJ := s[j]
57         if sI.GetPeriodID() != sJ.GetPeriodID() {
```

```
58                  return sI.GetPeriodID() < sJ.GetPeriodID()
59          }
60          if len(sI.GetID()) != len(sJ.GetID()) {
61                  return len(sI.GetID()) < len(sJ.GetID())
62          }
63          return sI.GetID() < sJ.GetID()
64  }
65
66  func (s sorter) Len() int {
67          return len(s)
68  }
```

Arquivo split.go

```
1   package ddiffer
2
3   import (
4           "fmt"
5           "io"
6
7           "github.com/fjorgemota/gurudamatricula/robot/format"
8           "github.com/fjorgemota/gurudamatricula/robot/format/pool"
9           "github.com/fjorgemota/gurudamatricula/util/coder"
10          "github.com/fjorgemota/gurudamatricula/util/file"
11          "github.com/pkg/errors"
12  )
13
14  type SplitterOptions struct {
15          Name           string
16          TargetTemplate string
17          Limits         Limit
18  }
19
20  type ConverterOptions struct {
21          UniversityID string
22          Target       TargetType
23  }
24
25  func getSplitOptions(options SplitterOptions) SplitterOptions {
26          if options.TargetTemplate == "" {
```

```
27                  options.TargetTemplate = options.Name + ".part%d"
28          }
29          return options
30  }
31
32  func splitBase(manager file.Manager, handler Handler, loader func(decoder
    ↪  coder.StreamingDecoder, encoder coder.StreamingEncoder, counter
    ↪  *counterWriter) error, options SplitterOptions) ([]string, error) {
33          var result []string
34          var err error
35          var reader io.ReadCloser
36          var decoder coder.StreamingDecoder
37          var counter *counterWriter
38          options = getSplitOptions(options)
39          reader, err = manager.Reader(options.Name)
40          if err == nil {
41                  decoder = handler.GetStreamingDecoder(reader)
42          }
43          var part int
44          encoder := coder.NewSplitterEncoder(func()
            ↪  (coder.StreamingEncoder, error) {
45                  tmpName := fmt.Sprintf(options.TargetTemplate, part)
46                  part++
47                  result = append(result, tmpName)
48                  writer, localErr := manager.Writer(tmpName)
49                  if localErr == nil {
50                          counter = &counterWriter{writer: writer, counter:
                            ↪  0}
51                  }
52                  var partEncoder coder.StreamingEncoder
53                  if localErr == nil {
54                          partEncoder =
                            ↪  handler.GetStreamingEncoder(counter)
55                  }
56                  return partEncoder, localErr
57          }, options.Limits.Items)
58          for err == nil {
59                  err = loader(decoder, encoder, counter)
60          }
```

```
61          if errors.Cause(err) == io.EOF {
62                  err = nil
63          }
64          if err == nil {
65                  err = encoder.Flush()
66          }
67          if err == nil {
68                  err = encoder.Close()
69          }
70          if err == nil {
71                  err = reader.Close()
72          }
73          return result, err
74  }
75
76  func split(manager file.Manager, handler Handler, loader func(load
    ↪  func(data interface{}) ([]string, error)) ([]string, error), clean
    ↪  func(data interface{}), options SplitterOptions) ([]string, error) {
77          return loader(func(data interface{}) ([]string, error) {
78                  return splitBase(manager, handler, func(decoder
                    ↪  coder.StreamingDecoder, encoder
                    ↪  coder.StreamingEncoder, counter *counterWriter) error
                    ↪  {
79                          clean(data)
80                          err := decoder.Decode(data)
81                          if err == nil {
82                                  err = encoder.Encode(data)
83                          }
84                          if err == nil && counter != nil &&
                            ↪  options.Limits.Size > 0 && counter.counter >
                            ↪  options.Limits.Size {
85                                  err = encoder.Close()
86                          }
87                          return err
88                  }, options)
89          })
90  }
91
```

```
92  func convertAndSplit(manager file.Manager, handler Handler, loader
    ↪  func(load func(data interface{}) ([]string, error)) ([]string, error),
    ↪  converter func(data interface{}, save Callback) error, clean
    ↪  func(data interface{}), options SplitterOptions) ([]string, error) {
93          return loader(func(data interface{}) ([]string, error) {
94                  return splitBase(manager, handler, func(decoder
                    ↪  coder.StreamingDecoder, encoder
                    ↪  coder.StreamingEncoder, counter *counterWriter) error
                    ↪  {
95                          clean(data)
96                          err := decoder.Decode(data)
97                          if err == nil {
98                                  err = converter(data, func(entity Entity)
                                    ↪  error {
99                                          err = entity.EncodeTo(encoder)
100                                         entity.Put()
101                                         if err == nil && counter != nil
                                            ↪  && options.Limits.Size > 0 &&
                                            ↪  counter.counter >
                                            ↪  options.Limits.Size {
102                                                 err = encoder.Close()
103                                         }
104                                         return err
105                                 })
106                         }
107                         return err
108                 }, options)
109         })
110 }
111
112 func teamHandler(load func(data interface{}) ([]string, error)) ([]string,
    ↪  error) {
113         team := pool.GetTeam()
114         result, err := load(team)
115         pool.PutTeam(team)
116         return result, err
117 }
118
119 func cleanTeamHandler(data interface{}) {
```

```
120         pool.CleanTeam(data.(*format.Team))
121 }
122
123 func habilitationHandler(load func(data interface{}) ([]string, error))
    ↪  ([]string, error) {
124         habilitation := pool.GetHabilitation()
125         result, err := load(habilitation)
126         pool.PutHabilitation(habilitation)
127         return result, err
128 }
129
130 func cleanHabilitationHandler(data interface{}) {
131         pool.CleanHabilitation(data.(*format.Habilitation))
132 }
133
134 func SplitSource(manager file.Manager, handler Handler, source SourceType,
    ↪  options SplitterOptions) ([]string, error) {
135         var handle func(load func(data interface{}) ([]string, error))
            ↪  ([]string, error)
136         var cleaner func(data interface{})
137         var err error
138         switch source {
139         case SourceOffers:
140                 handle = teamHandler
141                 cleaner = cleanTeamHandler
142         case SourceHabilitations:
143                 handle = habilitationHandler
144                 cleaner = cleanHabilitationHandler
145         default:
146                 err = errInvalidSource
147         }
148         var result []string
149         if err == nil {
150                 result, err = split(manager, handler, handle, cleaner,
                    ↪  options)
151         }
152         return result, err
153 }
154
```

```
155 func SplitTarget(manager file.Manager, handler Handler, target TargetType,
    ↪  options SplitterOptions) ([]string, error) {
156         var handle func(load func(data interface{}) ([]string, error))
            ↪  ([]string, error)
157         var cleaner func(data interface{})
158         getter, err := GetGetter(target)
159         if err == nil {
160                 handle = func(load func(data interface{}) ([]string,
                    ↪  error)) ([]string, error) {
161                         return load(getter())
162                 }
163                 cleaner = func(data interface{}) {
164                         entity := data.(Entity)
165                         entity.Clean()
166                 }
167         }
168         var result []string
169         if err == nil {
170                 result, err = split(manager, handler, handle, cleaner,
                    ↪  options)
171         }
172         return result, err
173 }
174
175 func Convert(manager file.Manager, handler Handler, source SourceType,
    ↪  splitOptions SplitterOptions, convertOptions ConverterOptions)
    ↪  ([]string, error) {
176         var handle func(load func(data interface{}) ([]string, error))
            ↪  ([]string, error)
177         var cleaner func(data interface{})
178         var convert func(data interface{}, save Callback) error
179         var err error
180         switch source {
181         case SourceOffers:
182                 handle = teamHandler
183                 cleaner = cleanTeamHandler
184                 var converter convertFromTeam
185                 converter, err = getTeamConversor(convertOptions.Target)
186                 convert = func(data interface{}, save Callback) error {
```

```
187                         return converter(convertOptions.UniversityID,
                            ↪  data.(*format.Team), save)
188                 }
189         case SourceHabilitations:
190                 handle = habilitationHandler
191                 cleaner = cleanHabilitationHandler
192                 var converter convertFromHabilitation
193                 converter, err =
                    ↪  getHabilitationConversor(convertOptions.Target)
194                 convert = func(data interface{}, save Callback) error {
195                         return converter(convertOptions.UniversityID,
                            ↪  data.(*format.Habilitation), save)
196                 }
197         default:
198                 err = errInvalidSource
199         }
200         var result []string
201         if err == nil {
202                 result, err = convertAndSplit(manager, handler, handle,
                    ↪  convert, cleaner, splitOptions)
203         }
204         return result, err
205 }
```

Arquivo teacher.go

```
1  package ddiffer
2
3  import (
4          "reflect"
5          "sync"
6
7          "github.com/fjorgemota/gurudamatricula/models/keygen"
8          "github.com/fjorgemota/gurudamatricula/robot/format"
9          "github.com/fjorgemota/gurudamatricula/util/coder"
10 )
11
12 type DiffTeacher struct {
13         ID           string         `json:"id"`
14         GenericID    string         `json:"generic_id"`
```

```
15          UniversityID string         `json:"university_id"`
16          Period       DiffForeignKey `json:"period_id"`
17          Teacher      format.Teacher `json:"teacher"`
18  }
19
20  var diffTeacherPool = sync.Pool{
21          New: func() interface{} {
22                  return new(DiffTeacher)
23          },
24  }
25
26  func (dt *DiffTeacher) GetID() string {
27          return dt.ID
28  }
29
30  func (dt *DiffTeacher) GetUniversityID() string {
31          return dt.UniversityID
32  }
33
34  func (dt *DiffTeacher) GetPeriodID() string {
35          return dt.Period.ID
36  }
37
38  func (dt *DiffTeacher) Equal(data Entity) bool {
39          teacher, ok := data.(*DiffTeacher)
40          return ok && reflect.DeepEqual(teacher.Teacher, dt.Teacher)
41  }
42
43  func (dt *DiffTeacher) Merge(data Entity) error {
44          return nil
45  }
46
47  func (dt *DiffTeacher) EncodeTo(encoder coder.StreamingEncoder) error {
48          return encoder.Encode(dt)
49  }
50
51  func (dt *DiffTeacher) DecodeFrom(decoder coder.StreamingDecoder) error {
52          dt.Clean()
53          return decoder.Decode(dt)
```

```
54 }
55
56 func (dt *DiffTeacher) Clean() {
57         *dt = DiffTeacher{}
58 }
59
60 func (dt *DiffTeacher) Put() {
61         dt.Clean()
62         diffTeacherPool.Put(dt)
63 }
64
65 func toDiffTeacher(universityID string, team *format.Team, save Callback)
   ↪  error {
66         periodID, err := keygen.GeneratePeriodID(universityID,
          ↪  team.Period)
67         periodRef := DiffForeignKey{
68                 ID:   periodID,
69                 Name: team.Period.Name,
70         }
71         for _, teacher := range team.Teachers {
72                 var id, genericID string
73                 if err == nil {
74                         id, err =
                          ↪  keygen.GenerateTeacherByPeriodID(universityID,
                          ↪  team.Period, teacher)
75                 }
76                 if err == nil {
77                         genericID, err =
                          ↪  keygen.GenerateTeacherID(universityID,
                          ↪  teacher)
78                 }
79                 if err == nil {
80                         result := diffTeacherPool.Get().(*DiffTeacher)
81                         result.ID = id
82                         result.GenericID = genericID
83                         result.UniversityID = universityID
84                         result.Period = periodRef
85                         result.Teacher = teacher
86                         err = save(result)
```

```
87                  }
88          }
89          return err
90  }
91
92  func GetDiffTeacher() Entity {
93          return diffTeacherPool.Get().(*DiffTeacher)
94  }
```

Arquivo team.go

```
1   package ddiffer
2
3   import (
4           "reflect"
5           "sync"
6
7           "github.com/fjorgemota/gurudamatricula/models/keygen"
8           "github.com/fjorgemota/gurudamatricula/robot/format"
9           "github.com/fjorgemota/gurudamatricula/robot/format/pool"
10          "github.com/fjorgemota/gurudamatricula/util/coder"
11  )
12
13  type DiffTeam struct {
14          ID                        string      `json:"id"`
15          PeriodID                  string      `json:"period_id"`
16          UniversityID              string      `json:"university_id"`
17          DisciplineOfferID         string      `json:"discipline_offer_id"`
18          DisciplineOfferGenericID string
            ↪  `json:"discipline_offer_generic_id"`
19          CourseID                  string      `json:"course_id"`
20          Team                      format.Team `json:"team"`
21  }
22
23  var diffTeamPool = sync.Pool{
24          New: func() interface{} {
25                  team := &DiffTeam{}
26                  pool.CleanTeam(&team.Team)
27                  return team
28          },
```

```
29 }
30
31 func (dt *DiffTeam) GetID() string {
32         return dt.ID
33 }
34
35 func (dt *DiffTeam) GetUniversityID() string {
36         return dt.UniversityID
37 }
38
39 func (dt *DiffTeam) GetPeriodID() string {
40         return dt.PeriodID
41 }
42
43 func (dt *DiffTeam) Equal(data Entity) bool {
44         team, ok := data.(*DiffTeam)
45         return ok && reflect.DeepEqual(team, dt)
46 }
47
48 func (dt *DiffTeam) Merge(data Entity) error {
49         return nil
50 }
51
52 func (dt *DiffTeam) EncodeTo(encoder coder.StreamingEncoder) error {
53         return encoder.Encode(dt)
54 }
55
56 func (dt *DiffTeam) DecodeFrom(decoder coder.StreamingDecoder) error {
57         dt.Clean()
58         return decoder.Decode(dt)
59 }
60
61 func (dt *DiffTeam) Clean() {
62         team := dt.Team
63         pool.CleanTeam(&team)
64         *dt = DiffTeam{
65                 Team: team,
66         }
67 }
```

```
68
69 func (dt *DiffTeam) Put() {
70         dt.Clean()
71         diffTeamPool.Put(dt)
72 }
73
74 func toDiffTeam(universityID string, team *format.Team, save Callback)
   ↪  error {
75         id, err := keygen.GenerateTeamID(universityID, *team)
76         var periodID, courseID, disciplineOfferID,
           ↪  disciplineOfferGenericID string
77         if err == nil {
78                 periodID, err = keygen.GeneratePeriodID(universityID,
                   ↪  team.Period)
79         }
80         if err == nil {
81                 disciplineOfferID, err =
                   ↪  keygen.GenerateDisciplineOfferID(universityID,
                   ↪  team.Period, team.Campus, team.Discipline)
82         }
83         if err == nil {
84                 disciplineOfferGenericID, err =
                   ↪  keygen.GenerateDisciplineGenericID(universityID,
                   ↪  team.Discipline.GenericID)
85         }
86         if err == nil && len(team.Course.ID) > 0 {
87                 courseID, err = keygen.GenerateCourseID(universityID,
                   ↪  format.Course{ID: team.Course.ID})
88         }
89         if err == nil {
90                 result := diffTeamPool.Get().(*DiffTeam)
91                 result.ID = id
92                 result.UniversityID = universityID
93                 result.PeriodID = periodID
94                 result.DisciplineOfferID = disciplineOfferID
95                 result.DisciplineOfferGenericID =
                   ↪  disciplineOfferGenericID
96                 result.CourseID = courseID
97                 pool.CopyTeam(&result.Team, team)
```

```
98                  err = save(result)
99          }
100         return err
101 }
102
103 func GetDiffTeam() Entity {
104         return diffTeamPool.Get().(*DiffTeam)
105 }
```

B.1.11 Pasta robot/fetcher

Arquivo fetcher.go

```
1  package fetcher
2
3  import (
4          "context"
5          "io"
6          "io/ioutil"
7          "net/http"
8          "net/url"
9          "sort"
10         "strconv"
11         "strings"
12         "time"
13
14         "github.com/qri-io/jsonschema"
15         "github.com/sirupsen/logrus"
16
17         "golang.org/x/crypto/bcrypt"
18
19         "github.com/fjorgemota/gurudamatricula/models"
20         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
21         "github.com/fjorgemota/gurudamatricula/robot/format"
22         "github.com/fjorgemota/gurudamatricula/robot/importer"
23         "github.com/fjorgemota/gurudamatricula/util/clock"
24         "github.com/fjorgemota/gurudamatricula/util/coder"
25         "github.com/fjorgemota/gurudamatricula/util/file"
26         "github.com/fjorgemota/gurudamatricula/util/kv"
27         "github.com/fjorgemota/gurudamatricula/util/pipeline"
```

```
28          uuid "github.com/gofrs/uuid"
29          "github.com/pkg/errors"
30  )
31
32  const secretCost = 15
33
34  type Handler interface {
35          GetStore(context.Context) (kv.Store, error)
36          GetManager(context.Context) (file.Manager, error)
37          GetHTTPClient(ctx context.Context, jar http.CookieJar)
            ↪  *http.Client
38          GetLogger(ctx context.Context) logrus.FieldLogger
39          GetStreamingDecoder(io.Reader) coder.StreamingDecoder
40          GetDDiffer() ddiffer.PipelineHandler
41          GetImporter() importer.PipelineHandler
42  }
43
44  type Tasks struct {
45          Lock              string
46          Monitor           string
47          Fetch             string
48          Validate          string
49          Prepare           string
50          Compare           string
51          Import            string
52          Optimize          string
53          DeleteIndexFiles string
54          Finalize          string
55  }
56
57  type Options struct {
58          Tasks                Tasks
59          Clock                clock.Clock
60          DelayCheck           time.Duration
61          RemoveFromQueueDelay time.Duration
62          LimitVersions        int
63          LimitQueue           int
64          Parallel             bool
65  }
```

```
66
67  type fetcher struct {
68          handler Handler
69          options Options
70  }
71
72  type Secret struct {
73          Public  string
74          Private string
75  }
76
77  func GenerateSecret() (Secret, error) {
78          var result Secret
79          var private []byte
80          id, err := uuid.NewV4()
81          if err == nil {
82                  result.Public = id.String()
83                  private, err =
                    ↪  bcrypt.GenerateFromPassword([]byte(result.Public),
                    ↪  secretCost)
84                  result.Public = "#" + result.Public
85          }
86          if err == nil {
87                  result.Private = "$" + string(private)
88          }
89          return result, err
90  }
91
92  func (f fetcher) Start(ctx context.Context, pipe pipeline.Pipeline,
    ↪  source ddiffer.SourceType, universityID, filename, secret string)
    ↪  error {
93          var university models.University
94          var store kv.Store
95          var id string
96          now := f.options.Clock.Now()
97          queueLimit := now.Add(f.options.RemoveFromQueueDelay)
98          idHandle, err := uuid.NewV4()
99          if err == nil {
100                 id = universityID + "-" + idHandle.String()
```

```
101         }
102         if err == nil {
103                 store, err = f.handler.GetStore(ctx)
104         }
105         if err == nil {
106                 refs := []kv.Reference{{Table: models.TableUniversity,
                    ↪ Key: universityID}}
107                 err = store.Transaction(refs, func(db kv.LimitedStore)
                    ↪ error {
108                         university = models.University{}
109                         localErr := db.Get(models.TableUniversity,
                            ↪ universityID, &university)
110                         var data *models.UniversityData
111                         switch source {
112                         case ddiffer.SourceOffers:
113                                 data = &university.Offers
114                         case ddiffer.SourceHabilitations:
115                                 data = &university.Habilitations
116                         }
117                         if localErr == nil &&
                            ↪ bcrypt.CompareHashAndPassword([]byte(data.Secret[1:]),
                            ↪ []byte(secret)) != nil {
118                                 localErr = errors.New("fetcher: Wrong
                                    ↪ secret")
119                         }
120                         newQueue := data.Queue[:0]
121                         for i := range data.Queue {
122                                 if
                                    ↪ data.Queue[i].LastUpdated.After(queueLimit)
                                    ↪ {
123                                         newQueue = append(newQueue,
                                            ↪ data.Queue[i])
124                                 }
125                         }
126                         data.Queue = newQueue
127                         if localErr == nil && len(data.Queue) <
                            ↪ f.options.LimitQueue {
128                                 data.Queue = append(data.Queue,
                                    ↪ models.UniversityQueueItem{
```

```
129                                         Filename:    filename,
130                                         ID:          id,
131                                         LastUpdated: now,
132                                 })
133                         } else if localErr == nil {
134                                 localErr = errors.New("fetcher: Too many
                                    ↪ requests")
135                         }
136                         if localErr == nil {
137                                 localErr = db.Set(models.TableUniversity,
                                    ↪ universityID, &university)
138                         }
139                         return localErr
140                 })
141         }
142         if kv.IsKeyNotFoundError(err) {
143                 err = errors.Wrap(err, "fetcher: University not
                    ↪ recognized")
144         }
145         if err == nil {
146                 _, err = pipe.Dispatch(f.options.Tasks.Lock, id,
                    ↪ universityID, source)
147         }
148         return err
149 }
150
151 func (f fetcher) lock(ctx context.Context, pipe pipeline.Pipeline, id
    ↪ string, universityID string, source ddiffer.SourceType) error {
152         var locked bool
153         var filename string
154         var university models.University
155         var data *models.UniversityData
156         var store kv.Store
157         var futID string
158         var found bool
159         logger := f.handler.GetLogger(ctx).WithField("id",
            ↪ id).WithField("university_id",
            ↪ university.ID).WithField("source", source)
160         logger.Info("Trying to lock..")
```

```
        futID, ok := pipeline.GetID(ctx)
        now := f.options.Clock.Now()
        queueLimit := now.Add(-f.options.RemoveFromQueueDelay)
        err := errors.New("Invalid context: cannot get pipeline ID")
        if ok {
                err = nil
        }
        if err == nil {
                store, err = f.handler.GetStore(ctx)
        }
        if err == nil {
                err = store.Transaction([]kv.Reference{{Table:
                ↪  models.TableUniversity, Key: universityID}}, func(db
                ↪  kv.LimitedStore) error {
                        university = models.University{}
                        localErr := db.Get(models.TableUniversity,
                        ↪  universityID, &university)
                        data = nil
                        switch source {
                        case ddiffer.SourceOffers:
                                data = &university.Offers
                        case ddiffer.SourceHabilitations:
                                data = &university.Habilitations
                        }
                        locked = false
                        filename = ""
                        newQueue := data.Queue[:0]
                        found = false
                        for i := range data.Queue {
                                if data.Queue[i].ID == id {
                                        data.Queue[i].LastUpdated = now
                                        found = true
                                }
                                if
                                ↪  data.Queue[i].LastUpdated.After(queueLimit)
                                ↪  {
                                        newQueue = append(newQueue,
                                        ↪  data.Queue[i])
                                }
```

```
194                         }
195                         data.Queue = newQueue
196                         if len(data.Queue) > 0 &&
                            ↪  data.Queue[len(data.Queue)-1].ID == id {
197                                 filename =
                                    ↪  data.Queue[len(data.Queue)-1].Filename
198                         }
199                         if localErr == nil && data.Mutex.Owner != "" &&
                            ↪  data.Mutex.LastUpdated.Before(queueLimit) {
200                                 data.Mutex.Owner = ""
201                         }
202                         if localErr == nil && filename != "" &&
                            ↪  data.Mutex.Owner == "" {
203                                 // Mutex has no owner! Profit!
204                                 data.Mutex.Owner = id
205                                 data.Mutex.LastUpdated = now
206                                 locked = true
207                         }
208                         if localErr == nil && locked {
209                                 // Only try to save if locked in
                                    ↪  successfully
210                                 localErr = db.Set(models.TableUniversity,
                                    ↪  universityID, &university)
211                         }
212                         return localErr
213                 })
214         }
215         if err == nil && found {
216                 if locked {
217                         logger.Info("Locked! Dispatching Fetch")
218                         _, err = pipe.Dispatch(f.options.Tasks.Fetch, id,
                            ↪  university, source, data.BaseURL, filename)
219                         if err == nil {
220                                 _, err =
                                    ↪  pipe.DispatchWithOptions(f.options.Tasks.Monito
                                    ↪  pipeline.TaskOptions{
221                                         Detach: true,
222                                         Delay:  f.options.DelayCheck,
223                                 }, id, futID, universityID, source)
```

```
224                         }
225                 } else if filename != "" {
226                         logger.Info("Already locked! Dispatching Lock
                            ↪  task..")
227                         _, err =
                            ↪  pipe.DispatchWithOptions(f.options.Tasks.Lock,
                            ↪  pipeline.TaskOptions{
228                             Delay: f.options.DelayCheck,
229                         }, id, universityID, source)
230                 }
231         }
232         return err
233 }
234
235 func (f fetcher) monitor(ctx context.Context, pipe pipeline.Pipeline, id,
    ↪  futID, universityID string, source ddiffer.SourceType) error {
236         var status pipeline.FutureStatus
237         logger := f.handler.GetLogger(ctx).WithField("id",
            ↪  id).WithField("university_id",
            ↪  universityID).WithField("source", source)
238         logger.Info("Monitoring task..")
239         store, err := f.handler.GetStore(ctx)
240         if err == nil {
241                 status, err = pipe.GetStatus(futID)
242         }
243         if err == nil && status == pipeline.Aborted {
244                 err = store.Transaction([]kv.Reference{{Table:
                    ↪  models.TableUniversity, Key: universityID}}, func(db
                    ↪  kv.LimitedStore) error {
245                         var university models.University
246                         localErr := db.Get(models.TableUniversity,
                            ↪  universityID, &university)
247                         saved := false
248                         if localErr == nil {
249                                 var data *models.UniversityData
250                                 switch source {
251                                 case ddiffer.SourceOffers:
252                                         data = &university.Offers
253                                 case ddiffer.SourceHabilitations:
```

```
254                                             data = &university.Habilitations
255                                     }
256                                     if data.Mutex.Owner == id {
257                                             // Clear mutex, because future
                                                ↪ was aborted
258                                             data.Mutex.Owner = ""
259                                             data.Mutex.LastUpdated =
                                                ↪ f.options.Clock.Now()
260                                             saved = true
261                                     }
262                             }
263                             if localErr == nil && saved {
264                                     // Only try to save if locked in
                                        ↪ successfully
265                                     localErr = db.Set(models.TableUniversity,
                                        ↪ universityID, &university)
266                             }
267                             return localErr
268                     })
269             }
270             if err == nil && status != pipeline.Done && status !=
                ↪ pipeline.Aborted {
271                     logger.Info("Task still running! Dispatching monitoring
                        ↪ task")
272                     _, err = pipe.DispatchWithOptions(f.options.Tasks.Monitor,
                        ↪ pipeline.TaskOptions{
273                             Delay: f.options.DelayCheck,
274                     }, id, futID, universityID, source)
275             } else if err == nil {
276                     logger.Info("Task terminated! Closing..")
277             } else {
278                     logger.WithError(err).Warn("Error while monitoring..")
279             }
280             return err
281     }
282
283     func (f fetcher) fetch(ctx context.Context, pipe pipeline.Pipeline, id
        ↪ string, university models.University, source ddiffer.SourceType,
        ↪ baseURL, downloadFilename string) error {
```

```
284         var response *http.Response
285         var writer io.WriteCloser
286         var manager file.Manager
287         var filename string
288         logger := f.handler.GetLogger(ctx).WithField("id",
            ↪  id).WithField("university_id",
            ↪  university.ID).WithField("source", source)
289         logger.Info("Fetching data..")
290         now := f.options.Clock.Now()
291         manager, err := f.handler.GetManager(ctx)
292         if err == nil {
293                 filename = id + "-" + strconv.FormatInt(now.Unix(), 10)
294                 writer, err = manager.Writer(filename)
295         }
296         if err == nil {
297                 url := strings.ReplaceAll(baseURL, "{filename}",
                    ↪  downloadFilename)
298                 logger.WithField("url", url).Info("Dispatching request")
299                 client := f.handler.GetHTTPClient(ctx, nil)
300                 response, err = client.Get(url)
301         }
302         if err == nil {
303                 _, err = io.Copy(writer, response.Body)
304         }
305         if err == nil {
306                 err = response.Body.Close()
307         }
308         if err == nil {
309                 err = writer.Close()
310         }
311         if err == nil {
312                 _, err = pipe.Dispatch(f.options.Tasks.Validate, id,
                    ↪  filename, university, source)
313         }
314         return err
315 }
316
```

```
317 func (f fetcher) validate(ctx context.Context, pipe pipeline.Pipeline, id,
    ↪ filename string, university models.University, source
    ↪ ddiffer.SourceType) error {
318         var reader io.ReadCloser
319         var decoder coder.StreamingDecoder
320         logger := f.handler.GetLogger(ctx).WithField("id",
            ↪ id).WithField("university_id",
            ↪ university.ID).WithField("source", source)
321         manager, err := f.handler.GetManager(ctx)
322         if err == nil {
323                 reader, err = manager.Reader(filename)
324         }
325         if err == nil {
326                 decoder = coder.NewJSONNativeDecoder(reader)
327         }
328         valid := true
329         for err == nil && valid {
330                 var result []jsonschema.ValError
331                 if source == ddiffer.SourceHabilitations {
332                         result, err =
                            ↪ format.ValidateHabilitation(decoder)
333                 } else if source == ddiffer.SourceOffers {
334                         result, err = format.ValidateTeam(decoder)
335                 } else {
336                         valid = false
337                 }
338                 if err == nil && result != nil && valid {
339                         valid = len(result) == 0
340                 }
341                 if err == nil && !valid && result != nil {
342                         for _, resultError := range result {
343                                 logger.Errorf("Found validation error:
                                    ↪ %s", resultError.Error())
344                         }
345                 }
346         }
347         if errors.Cause(err) == io.EOF {
348                 err = nil
349         }
```

```
350         if err == nil {
351                 if valid {
352                         logger.Info("File validated. Dispatching convert
                            ↪  task..")
353                         var convertResult pipeline.FutureID
354                         convertResult, err =
                            ↪  f.handler.GetDDiffer().Convert(pipe,
                            ↪  ddiffer.PipelineConverterOptions{
355                                 Name:           filename,
356                                 Source:         source,
357                                 TargetTemplate: filename + ".%s",
358                                 UniversityID:   university.ID,
359                         })
360                         if err == nil {
361                                 _, err =
                                    ↪  pipe.Dispatch(f.options.Tasks.Prepare,
                                    ↪  id, filename, university, source,
                                    ↪  convertResult)
362                         }
363                 } else {
364                         logger.Error("File is not valid")
365                 }
366         }
367         return err
368 }
369
370 func (f fetcher) prepare(ctx context.Context, pipe pipeline.Pipeline, id,
    ↪  filename string, university models.University, source
    ↪  ddiffer.SourceType, results []ddiffer.PipelineConverterResult) error
    ↪  {
371         var archivedPeriods []string
372         var zero struct{}
373         manager, err := f.handler.GetManager(ctx)
374         if source == ddiffer.SourceOffers {
375                 archivedPeriodsMap := make(map[string]struct{},
                    ↪  len(university.Periods))
376                 seen := make(map[string]struct{},
                    ↪  len(university.Periods))
377                 for _, period := range university.Periods {
```

```
378                     archivedPeriodsMap[period.ID] = zero
379                     seen[period.ID] = zero
380             }
381             var newPeriods []models.ForeignKey
382             periods := make([]models.ForeignKey,
                ↪ len(university.Periods))
383             copy(periods, university.Periods)
384             var filename string
385             for _, result := range results {
386                     if result.Target == ddiffer.TargetPeriod {
387                             filename = result.Result
388                             break
389                     }
390             }
391             var reader io.ReadCloser
392             var getter ddiffer.Getter
393             if err == nil && filename != "" {
394                     getter, err =
                        ↪ ddiffer.GetGetter(ddiffer.TargetPeriod)
395             }
396             if err == nil && filename != "" {
397                     reader, err = manager.Reader(filename)
398             }
399             var dec coder.StreamingDecoder
400             if err == nil && filename != "" {
401                     dec = f.handler.GetStreamingDecoder(reader)
402             }
403             if err == nil && filename != "" {
404                     row := getter().(*ddiffer.DiffPeriod)
405                     for err == nil {
406                             row.Clean()
407                             err = row.DecodeFrom(dec)
408                             if err == nil {
409                                     periodID := row.ID
410                                     if _, saw := seen[periodID]; !saw
                                        ↪ {
411                                             newPeriods =
                                                ↪ append(newPeriods,
                                                ↪ models.ForeignKey{
```

```
412                                                     ID:   periodID,
413                                                     Name:
                                                        ↪  row.Period.Name,
414                                             })
415                                             seen[periodID] = zero
416                                     }
417                                     delete(archivedPeriodsMap,
                                        ↪  periodID)
418                             }
419                     }
420                     if errors.Cause(err) == io.EOF {
421                             err = nil
422                             row.Clean()
423                             row.Put()
424                     }
425             }
426             if err == nil && filename != "" {
427                     err = reader.Close()
428             }
429             archivedPeriods = make([]string, 0,
                ↪  len(archivedPeriodsMap))
430             for archivedPeriod := range archivedPeriodsMap {
431                     archivedPeriods = append(archivedPeriods,
                        ↪  archivedPeriod)
432             }
433             if err == nil {
434                     periods = append(periods, newPeriods...)
435                     sort.Sort(foreignKeysByName(periods))
436                     university.Periods = periods
437             }
438     } else if source == ddiffer.SourceHabilitations {
439             seen := make(map[string]struct{})
440             var courses []models.ForeignKey
441             var file string
442             for _, result := range results {
443                     if result.Target == ddiffer.TargetCourse {
444                             file = result.Result
445                             break
446                     }
```

```
447                 }
448                 var reader io.ReadCloser
449                 var getter ddiffer.Getter
450                 if err == nil && file != "" {
451                         getter, err =
                        ↪  ddiffer.GetGetter(ddiffer.TargetCourse)
452                 }
453                 if err == nil && file != "" {
454                         reader, err = manager.Reader(file)
455                 }
456                 var dec coder.StreamingDecoder
457                 if err == nil && file != "" {
458                         dec = f.handler.GetStreamingDecoder(reader)
459                 }
460                 if err == nil && file != "" {
461                         row := getter().(*ddiffer.DiffCourse)
462                         for err == nil {
463                                 row.Clean()
464                                 err = row.DecodeFrom(dec)
465                                 if err == nil {
466                                         id := row.GetID()
467                                         if _, ok := seen[id]; !ok {
468                                                 seen[id] = zero
469                                                 courses = append(courses,
                                                ↪  models.ForeignKey{
470                                                         ID:   id,
471                                                         Name:
                                                        ↪  row.Course.Name,
472                                                 })
473                                         }
474                                 }
475                         }
476                         if errors.Cause(err) == io.EOF {
477                                 err = nil
478                                 row.Clean()
479                                 row.Put()
480                         }
481                 }
482                 if err == nil && file != "" {
```

```
                        err = reader.Close()
                }
                if err == nil {
                        sort.Sort(foreignKeysByName(courses))
                        university.Courses = courses
                }
        }
        if err == nil {
                if f.options.Parallel {
                        var importResults []pipeline.FutureID
                        for _, result := range results {
                                var compareResult pipeline.FutureID
                                if err == nil {
                                        compareResult, err =
                                        ↪ f.compareInternal(pipe, id,
                                        ↪ university, archivedPeriods,
                                        ↪ result)
                                }
                                var importResult []pipeline.FutureID
                                if err == nil {
                                        importResult, err =
                                        ↪ pipe.Dispatch(f.options.Tasks.Import,
                                        ↪ id, compareResult)
                                }
                                if err == nil {
                                        importResults =
                                        ↪ append(importResults,
                                        ↪ importResult[0])
                                        if
                                        ↪ importer.HasIndex(result.Target)
                                        ↪ {
                                                // We updated the index
                                                ↪ on importData!
                                                // So here we will
                                                ↪ concurrently optimize
                                                ↪ the index..while
                                                ↪ updating or even
                                                // finalizing everything
```

```
508                                                     _, err =
                                                        ↪  pipe.Dispatch(f.options.Tasks.O
                                                        ↪  id, importResult[0])
509                                             }
510                                     }
511                             }
512                             if err == nil {
513                                     args := make([]interface{},
                                        ↪  len(importResults)+4)
514                                     args[0] = id
515                                     args[1] = filename
516                                     args[2] = university
517                                     args[3] = source
518                                     for i, result := range importResults {
519                                             // We don't really need these
                                                ↪  results BUT
520                                             // we pass them to finalize
                                                ↪  because we need to await to
                                                ↪  finalize imports before
521                                             // changing main record
522                                             args[i+4] = result
523                                     }
524                                     _, err =
                                        ↪  pipe.Dispatch(f.options.Tasks.Finalize,
                                        ↪  args...)
525                             }
526                     } else {
527                             _, err = pipe.Dispatch(f.options.Tasks.Compare,
                                ↪  id, filename, university, source,
                                ↪  archivedPeriods, results)
528                     }
529             }
530             return err
531     }
532
533     func (f fetcher) compareInternal(pipe pipeline.Pipeline, id string,
        ↪  university models.University, archivedPeriods []string,
        ↪  converterResult ddiffer.PipelineConverterResult) (pipeline.FutureID,
        ↪  error) {
```

```
534         targetType := converterResult.Target.String()
535         lastFile :=
            ↪ university.GetLatestFile(f.getTarget(converterResult.Target))
536         var partTemplate string
537         if lastFile.File == "" {
538                 partTemplate = id + "-" + strings.ToLower(targetType) +
                    ↪ "-none-" + converterResult.Result
539         } else {
540                 partTemplate = id + "-" + strings.ToLower(targetType) +
                    ↪ "-" + lastFile.File + "-" + converterResult.Result
541         }
542         return f.handler.GetDDiffer().Compare(pipe,
            ↪ ddiffer.ComparerOptions{
543                 ArchivedPeriods:  archivedPeriods,
544                 OldSource:        lastFile.File,
545                 NewSource:        converterResult.Result,
546                 Target:           converterResult.Target,
547                 CreatedTemplate:  "created-part-%d-" + partTemplate,
548                 ModifiedTemplate: "modified-part-%d" + partTemplate,
549                 DeletedTemplate:  "deleted-part-%d-" + partTemplate,
550         })
551 }
552
553 func (f fetcher) compare(ctx context.Context, pipe pipeline.Pipeline, id,
    ↪ filename string, university models.University, source
    ↪ ddiffer.SourceType, archivedPeriods []string, converterResults
    ↪ []ddiffer.PipelineConverterResult, importResults
    ↪ ...ddiffer.ComparerResult) error {
554         var converterResult ddiffer.PipelineConverterResult
555         var err error
556         converterResult, converterResults =
            ↪ converterResults[len(converterResults)-1],
            ↪ converterResults[:len(converterResults)-1]
557         var compareResult pipeline.FutureID
558         if err == nil {
559                 compareResult, err = f.compareInternal(pipe, id,
                    ↪ university, archivedPeriods, converterResult)
560         }
561         count := 5
```

```
562         if len(converterResults) > 0 {
563                 count += 2
564         }
565         newArgs := make([]interface{}, len(importResults)+count)
566         newArgs[0] = id
567         newArgs[1] = filename
568         newArgs[2] = university
569         newArgs[3] = source
570         if len(converterResults) > 0 {
571                 newArgs[4] = archivedPeriods
572                 newArgs[5] = converterResults
573         }
574         for i, importResultInput := range importResults {
575                 newArgs[count+i] = importResultInput
576         }
577         var importResult []pipeline.FutureID
578         if err == nil {
579                 importResult, err = pipe.Dispatch(f.options.Tasks.Import,
                    ↪  id, compareResult)
580         }
581         if err == nil && importer.HasIndex(converterResult.Target) {
582                 // We updated the index on importData!
583                 // So here we will concurrently optimize the index..while
                    ↪  updating or even
584                 // finalizing everything
585                 importResult, err =
                    ↪  pipe.Dispatch(f.options.Tasks.Optimize, id,
                    ↪  importResult[0])
586         }
587         if err == nil {
588                 newArgs[count-1] = importResult[0]
589         }
590         if len(converterResults) == 0 {
591                 _, err = pipe.Dispatch(f.options.Tasks.Finalize,
                    ↪  newArgs...)
592         } else {
593                 _, err = pipe.Dispatch(f.options.Tasks.Compare,
                    ↪  newArgs...)
594         }
```

```
595         return err
596 }
597 func (f fetcher) importData(ctx context.Context, pipe pipeline.Pipeline,
    ↪  id string, result ddiffer.ComparerResult) (ddiffer.ComparerResult,
    ↪  error) {
598         var err error
599         if !ddiffer.IsComparerResultEmpty(result) &&
            ↪  (importer.HasDataset(result.Target) ||
            ↪  importer.HasIndex(result.Target)) {
600                 _, err = f.handler.GetImporter().Import(pipe, id, result)
601         }
602         return result, err
603 }
604
605 func (f fetcher) optimizeIndex(ctx context.Context, pipe
    ↪  pipeline.Pipeline, id string, result ddiffer.ComparerResult)
    ↪  (ddiffer.ComparerResult, error) {
606         var err error
607         if !ddiffer.IsComparerResultEmpty(result) {
608                 var futID pipeline.FutureID
609                 futID, err = f.handler.GetImporter().OptimizeIndex(pipe,
                    ↪  id, result)
610                 if err == nil {
611                         _, err =
                            ↪  pipe.Dispatch(f.options.Tasks.DeleteIndexFiles,
                            ↪  id, futID)
612                 }
613         }
614         return result, err
615 }
616
617 func (f fetcher) deleteIndexFiles(ctx context.Context, pipe
    ↪  pipeline.Pipeline, id string, result ddiffer.ComparerResult) error {
618         manager, err := f.handler.GetManager(ctx)
619         if err == nil {
620                 logger := f.handler.GetLogger(ctx).WithField("id",
                    ↪  id).WithField("target", result.Target)
621                 var files []string
622                 files = append(files, result.Created...)
```

```
623                 files = append(files, result.Modified...)
624                 files = append(files, result.Deleted...)
625                 logger.Debugf("Deleting %d files", len(files))
626                 err = file.DeleteFiles(manager, files)
627         }
628         return err
629 }
630
631 func (f fetcher) getTarget(target ddiffer.TargetType) string {
632         result := target.String()
633         return "target/" + result
634 }
635
636 func (f fetcher) getOriginal(source ddiffer.SourceType) string {
637         result := source.String()
638         return "original/" + result
639 }
640
641 func (f fetcher) finalize(ctx context.Context, pipe pipeline.Pipeline, id,
    ↪  filename string, university models.University, source
    ↪  ddiffer.SourceType, compareResults ...ddiffer.ComparerResult) error {
642         logger := f.handler.GetLogger(ctx).WithField("id",
           ↪  id).WithField("university_id",
           ↪  university.ID).WithField("source", source)
643         logger.Info("Finalizing")
644         now := f.options.Clock.Now()
645         store, err := f.handler.GetStore(ctx)
646         var manager file.Manager
647         var data, universityData *models.UniversityData
648         var toDelete, toDeleteFilenames []string
649         var deleteURL string
650         if err == nil {
651                 manager, err = f.handler.GetManager(ctx)
652         }
653         if err == nil {
654                 err = store.Transaction([]kv.Reference{{Table:
                   ↪  models.TableUniversity, Key: university.ID}}, func(db
                   ↪  kv.LimitedStore) error {
655                         var entity models.University
```

```
656                     localErr := db.Get(models.TableUniversity,
                        ↪  university.ID, &entity)
657                     if localErr == nil {
658                             equal := false
659                             switch source {
660                             case ddiffer.SourceOffers:
661                                     data = &entity.Offers
662                                     universityData =
                                        ↪  &university.Offers
663                             case ddiffer.SourceHabilitations:
664                                     data = &entity.Habilitations
665                                     universityData =
                                        ↪  &university.Habilitations
666                             }
667                             equal = data.Mutex.Owner ==
                                ↪  universityData.Mutex.Owner
668                             deleteURL = data.DeleteURL
669                             localErr = errors.New("Unexpected error:
                                ↪  change in mutex detected while
                                ↪  finalizing")
670                             if equal {
671                                     localErr = nil
672                             }
673                     }
674                     if localErr == nil {
675                             for i, result := range data.Queue {
676                                     toDeleteFilenames =
                                        ↪  append(toDeleteFilenames,
                                        ↪  result.Filename)
677                                     if result.ID == id {
678                                             data.Queue =
                                                ↪  data.Queue[i+1:]
679                                             break
680                                     }
681                             }
682                             for _, result := range compareResults {
683                                     target :=
                                        ↪  f.getTarget(result.Target)
```

```
684                                     entity.AddFile(target,
                                        ↪ result.NewSource)
685                                 }
686                                 original := f.getOriginal(source)
687                                 entity.AddFile(original, filename)
688                                 oldVersions := make(map[string]int,
                                    ↪ len(compareResults)+1)
689                                 for _, result := range compareResults {
690                                     target := result.Target.String()
691                                     latestVersion :=
                                        ↪ entity.GetLatestFile(target).Version
692                                     oldVersions[target] =
                                        ↪ latestVersion -
                                        ↪ f.options.LimitVersions
693                                 }
694                                 oldVersions[original] =
                                    ↪ entity.GetLatestFile(original).Version
                                    ↪ - f.options.LimitVersions
695                                 newFiles := make([]models.UniversityFile,
                                    ↪ 0, len(entity.Files))
696                                 toDelete = toDelete[:0]
697                                 for _, file := range entity.Files {
698                                     limit, ok := oldVersions[file.ID]
699                                     if !ok || (ok && file.Version >
                                        ↪ limit) {
700                                         newFiles =
                                            ↪ append(newFiles,
                                            ↪ file)
701                                     } else if ok {
702                                         toDelete =
                                            ↪ append(toDelete,
                                            ↪ file.File)
703                                     }
704                                 }
705                                 entity.Files = newFiles
706                                 switch source {
707                                 case ddiffer.SourceOffers:
708                                     entity.Periods =
                                        ↪ university.Periods
```

```
                                case ddiffer.SourceHabilitations:
                                        entity.Courses =
                                        ↪  university.Courses
                                }
                                data.LastUpdated = now
                        }
                        if localErr == nil {
                                localErr = db.Set(models.TableUniversity,
                                ↪  university.ID, &entity)
                        }
                        return localErr
                })
        }
        if err == nil {
                for _, result := range compareResults {
                        if importer.HasIndex(result.Target) {
                                // Files that have indexes will be
                                ↪  processed by optimize index instead
                                continue
                        }
                        toDelete = append(toDelete, result.Created...)
                        toDelete = append(toDelete, result.Modified...)
                        toDelete = append(toDelete, result.Deleted...)
                }
        }
        if err == nil && len(toDelete) > 0 {
                logger.Debugf("Deleting %d files", len(toDelete))
                // Clear old files :D
                err = file.DeleteFiles(manager, toDelete)
        }
        if len(deleteURL) > 0 {
                client := f.handler.GetHTTPClient(ctx, nil)
                var to *url.URL
                var response *http.Response
                if err == nil {
                        to, err = url.Parse(deleteURL)
                }
                if err == nil {
                        query := to.Query()
```

```
745                         query["filenames"] = toDeleteFilenames
746                         to.RawQuery = query.Encode()
747                         response, err = client.Get(to.String())
748                 }
749                 if err == nil {
750                         _, err = io.Copy(ioutil.Discard, response.Body)
751                 }
752                 if err == nil {
753                         err = response.Body.Close()
754                 }
755         }
756         return err
757 }
758 func getDefaultOptions(opt Options) Options {
759         if opt.Tasks.Lock == "" {
760                 opt.Tasks.Lock = "fetcher.internal.lock"
761         }
762         if opt.Tasks.Monitor == "" {
763                 opt.Tasks.Monitor = "fetcher.internal.monitor"
764         }
765         if opt.Tasks.Fetch == "" {
766                 opt.Tasks.Fetch = "fetcher.internal.fetch"
767         }
768         if opt.Tasks.Validate == "" {
769                 opt.Tasks.Validate = "fetcher.internal.validate"
770         }
771         if opt.Tasks.Prepare == "" {
772                 opt.Tasks.Prepare = "fetcher.internal.prepare"
773         }
774         if opt.Tasks.Compare == "" {
775                 opt.Tasks.Compare = "fetcher.internal.compare"
776         }
777         if opt.Tasks.Import == "" {
778                 opt.Tasks.Import = "fetcher.internal.import"
779         }
780         if opt.Tasks.Optimize == "" {
781                 opt.Tasks.Optimize = "fetcher.internal.optimize"
782         }
783         if opt.Tasks.DeleteIndexFiles == "" {
```

```
784                 opt.Tasks.DeleteIndexFiles =
                    ↪  "fetcher.internal.delete_index_files"
785         }
786         if opt.Tasks.Finalize == "" {
787                 opt.Tasks.Finalize = "fetcher.internal.finalize"
788         }
789         if opt.Clock == nil {
790                 opt.Clock = clock.NewRealClock()
791         }
792         if opt.DelayCheck < time.Minute {
793                 // Minimum of 1 minute
794                 opt.DelayCheck = time.Minute
795         }
796         if opt.RemoveFromQueueDelay < time.Minute {
797                 opt.RemoveFromQueueDelay = opt.DelayCheck
798         }
799         if opt.LimitVersions < 2 {
800                 opt.LimitVersions = 2
801         }
802         if opt.LimitQueue < 2 {
803                 opt.LimitQueue = 2
804         }
805         return opt
806 }
807
808 type Fetcher interface {
809         Start(ctx context.Context, pipe pipeline.Pipeline, source
          ↪  ddiffer.SourceType, universityID, filename, secret string)
          ↪  error
810 }
811
812 func NewFetcher(disp pipeline.Dispatcher, handler Handler, opt Options)
    ↪  (Fetcher, error) {
813         var err error
814         opt = getDefaultOptions(opt)
815         instance := fetcher{
816                 handler: handler,
817                 options: opt,
818         }
```

```
819         err = disp.Register(opt.Tasks.Lock, instance.lock)
820         if err == nil {
821                 err = disp.Register(opt.Tasks.Monitor, instance.monitor)
822         }
823         if err == nil {
824                 err = disp.Register(opt.Tasks.Fetch, instance.fetch)
825         }
826         if err == nil {
827                 err = disp.Register(opt.Tasks.Validate,
                    ↪  instance.validate)
828         }
829         if err == nil {
830                 err = disp.Register(opt.Tasks.Prepare, instance.prepare)
831         }
832         if err == nil {
833                 err = disp.Register(opt.Tasks.Compare, instance.compare)
834         }
835         if err == nil {
836                 err = disp.Register(opt.Tasks.Import,
                    ↪  instance.importData)
837         }
838         if err == nil {
839                 err = disp.Register(opt.Tasks.Optimize,
                    ↪  instance.optimizeIndex)
840         }
841         if err == nil {
842                 err = disp.Register(opt.Tasks.DeleteIndexFiles,
                    ↪  instance.deleteIndexFiles)
843         }
844         if err == nil {
845                 err = disp.Register(opt.Tasks.Finalize,
                    ↪  instance.finalize)
846         }
847         return instance, err
848 }
```

Arquivo sorter.go

```
1  package fetcher
2
```

```
3  import "github.com/fjorgemota/gurudamatricula/models"
4
5  type foreignKeysByName []models.ForeignKey
6
7  func (up foreignKeysByName) Len() int {
8          return len(up)
9  }
10
11 func (up foreignKeysByName) Less(i, j int) bool {
12         return up[i].Name < up[j].Name
13 }
14
15 func (up foreignKeysByName) Swap(i, j int) {
16         up[i], up[j] = up[j], up[i]
17 }
```

B.1.12 Pasta robot/format

Arquivo base.go

```
1  package format
2
3  import (
4          "encoding/gob"
5          "encoding/json"
6          "net/http"
7
8          "github.com/qri-io/jsonschema"
9
10         "github.com/gobuffalo/packr/v2"
11 )
12
13 //go:generate codecgen -o $GOPACKAGE.generated.go offers.go semantic.go
   ↪  table.go
14 //go:generate packr2
15
16 var teamSchema, habilitationSchema *jsonschema.RootSchema
17 var dataFileSystem *packr.Box
18 var globalErr error
19
```

```
20 func initGOB() {
21         gob.Register(Exclusivity(""))
22         // Tables
23         gob.Register(Column{})
24         gob.Register(Table{})
25         // Offers
26         gob.Register(Vacancy{})
27         gob.Register(Period{})
28         gob.Register(HourMinute{})
29         gob.Register(Schedule{})
30         gob.Register(Teacher{})
31         gob.Register(DisciplineOffer{})
32         gob.Register(Campus{})
33         gob.Register([]Campus{})
34         gob.Register(Team{})
35         // Semantic
36         gob.Register(DisciplineRelated{})
37         gob.Register(DisciplineWorkload{})
38         gob.Register(Discipline{})
39         gob.Register(Step{})
40         gob.Register(Limit{})
41         gob.Register(Habilitation{})
42         gob.Register(Course{})
43 }
44
45 func loadSchema(box *packr.Box, name string) (*jsonschema.RootSchema,
   ↪  error) {
46         var schema *jsonschema.RootSchema
47         data, err := box.Find(name)
48         if err == nil {
49                 schema = &jsonschema.RootSchema{}
50                 err = json.Unmarshal(data, schema)
51         }
52         return schema, err
53 }
54
55 func initSchemas() {
56         dataFileSystem = packr.New("data", "./data")
```

```
57          habilitationSchema, globalErr = loadSchema(dataFileSystem,
            ↪  "schema.habilitation.json")
58          if globalErr == nil {
59                  teamSchema, globalErr = loadSchema(dataFileSystem,
                    ↪  "schema.team.json")
60          }
61  }
62
63  func GetFileSystem() http.FileSystem {
64          return dataFileSystem
65  }
66
67  func init() {
68          initGOB()
69          initSchemas()
70  }
```

Arquivo offers.go

```
1   package format
2
3   type Exclusivity string
4
5   const (
6           Exclusive    Exclusivity = "yes"
7           NotExclusive Exclusivity = "no"
8           Unknown      Exclusivity = "unknown"
9   )
10
11  // Vacancy represents a table with data about vacancies
12  type Vacancy struct {
13          Offered int `json:"offered"`
14          Filled  int `json:"filled"`
15  }
16
17  // Period represents a semester of the university
18  type Period struct {
19          ID   string `json:"id"`
20          Name string `json:"name"`
21  }
```

```
22
23 // HourMinute defines an hour to define hour and minute
24 type HourMinute struct {
25         Hour   int8 `json:"hour"`
26         Minute int8 `json:"minute"`
27 }
28
29 // Room is where a class occurs, at a determined schedule
30 type Room struct {
31         ID          string `json:"id"`
32         Name        string `json:"name"`
33         OLCode      string `json:"olcode,omitempty"`
34         Description string `json:"description,omitempty"`
35 }
36
37 // Schedule represents a schedule of the university
38 type Schedule struct {
39         Start     HourMinute `json:"start"`
40         End       HourMinute `json:"end"`
41         DayOfWeek int8       `json:"dayOfWeek"`
42         Room      Room       `json:"room,omitempty"`
43 }
44
45 // Teacher represents a teacher of the university
46 type Teacher struct {
47         ID   string `json:"id"`
48         Name string `json:"name"`
49         URL  string `json:"url,omitempty"`
50 }
51
52 // CourseOffer represents a course in the university associated with the
   ↪  team
53 type CourseOffer struct {
54         ID        string      `json:"id"`
55         Name      string      `json:"name"`
56         Exclusive Exclusivity `json:"exclusive,omitempty"`
57 }
58
59 // DisciplineOffer represents a discipline of the university
```

```go
type DisciplineOffer struct {
        ID        string `json:"id"`
        GenericID string `json:"generic_id"`
        Code      string `json:"code"`
        Name      string `json:"name"`
}

// Campus represents a campus of the university
type Campus struct {
        ID     string `json:"id"`
        Name   string `json:"name"`
        OLCode string `json:"olcode,omitempty"`
}

// Team represents a team of the university
type Team struct {
        ID         string          `json:"id"`
        Code       string          `json:"code"`
        Schedules  []Schedule      `json:"schedules,omitempty"`
        Teachers   []Teacher       `json:"teachers,omitempty"`
        Vacancies  Vacancy         `json:"vacancies"`
        Campus     Campus          `json:"campus"`
        Period     Period          `json:"period"`
        Discipline DisciplineOffer `json:"discipline"`
        Course     CourseOffer     `json:"course,omitempty"`
        Tables     []Table         `json:"tables,omitempty"`
}
```

Arquivo semantic.go

```go
package format

// DisciplineRelatedItem is a discipline or text related to another
↪  discipline
type DisciplineRelatedItem struct {
        Type        string `json:"type"`
        Value       string `json:"value"`
        Description string `json:"description,omitempty"`
}

```

```
// DisciplineRelated represents a set of disciplines or text related to another
// discipline
type DisciplineRelated struct {
        Name  string                 `json:"name"`
        Type  string                 `json:"type"`
        Items []DisciplineRelatedItem `json:"items"`
}

// DisciplineWorkload defines a workload related to a discipline
type DisciplineWorkload struct {
        ID    string  `json:"id"`
        Name  string  `json:"name"`
        Unit  string  `json:"unit"`
        Value float64 `json:"value"`
}

// Discipline is a discipline of a habilitation of a course in a university
type Discipline struct {
        ID          string              `json:"id"`
        GenericID   string              `json:"generic_id"`
        Code        string              `json:"code"`
        Name        string              `json:"name"`
        Description string              `json:"description,omitempty"`
        Type        string              `json:"type"`
        Workload    []DisciplineWorkload `json:"workload,omitempty"`
        Related     []DisciplineRelated  `json:"related,omitempty"`
        Tables      []Table              `json:"tables,omitempty"`
}

// Step is a set of disciplines that can be done on one period of a habilitation
type Step struct {
        ID          string       `json:"id"`
        Name        string       `json:"name"`
        Disciplines []Discipline `json:"disciplines"`
}

```

```
46 // Limit defines a limit related to a habilitation
47 type Limit struct {
48         ID      string             `json:"id"`
49         Name    string             `json:"name"`
50         Context string             `json:"context"`
51         Unit    string             `json:"unit"`
52         Values  map[string]float64 `json:"values"`
53 }
54
55 // Course represents a course in a university
56 type Course struct {
57         ID      string `json:"id"`
58         Name    string `json:"name"`
59         URL     string `json:"url"`
60         Version string `json:"version"`
61 }
62
63 // Habilitation is a habilitation of a course in a university
64 type Habilitation struct {
65         ID          string  `json:"id"`
66         Name        string  `json:"name"`
67         Observation string  `json:"observation,omitempty"`
68         Limits      []Limit `json:"limits,omitempty"`
69         Tables      []Table `json:"tables,omitempty"`
70         Steps       []Step  `json:"steps"`
71         Course      Course  `json:"course"`
72 }
```

Arquivo table.go

```
1 package format
2
3 // Column represents a table
4 type Column struct {
5         ID   string `json:"id"`
6         Name string `json:"name,omitempty"`
7 }
8
9 // Table represents a generic table structure
10 type Table struct {
```

```
        ID          string              `json:"id"`
        Title       string              `json:"title"`
        Description string              `json:"description,omitempty"`
        Columns     []Column            `json:"columns"`
        Rows        []map[string]string `json:"rows,omitempty"`
}
```

Arquivo validator.go

```
package format

import (
        "github.com/fjorgemota/gurudamatricula/util/coder"
        "github.com/qri-io/jsonschema"
)

func validate(schema *jsonschema.RootSchema, decoder
↪  coder.StreamingDecoder) ([]jsonschema.ValError, error) {
        var result []jsonschema.ValError
        var value interface{}
        err := globalErr
        if err == nil {
                err = decoder.Decode(&value)
        }
        if err == nil {
                schema.Validate("/", value, &result)
        }
        return result, err
}

func ValidateHabilitation(decoder coder.StreamingDecoder)
↪  ([]jsonschema.ValError, error) {
        return validate(habilitationSchema, decoder)
}

func ValidateTeam(decoder coder.StreamingDecoder) ([]jsonschema.ValError,
↪  error) {
        return validate(teamSchema, decoder)
}
```

B.1.13 Pasta robot/importer

Arquivo base.go

```
1  package importer
2
3  import (
4          "io"
5
6          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
7          "github.com/fjorgemota/gurudamatricula/util/coder"
8          "github.com/fjorgemota/gurudamatricula/util/indexer"
9          "github.com/fjorgemota/gurudamatricula/util/kv"
10         "github.com/pkg/errors"
11 )
12
13 const temporaryTable = "index"
14
15 type IndexData struct {
16         Target   ddiffer.TargetType
17         ParentID string
18 }
19
20 type callback func(entity ddiffer.Entity, version string, read func(data
   ↪  interface{}) error, save func(id string, data interface{}) error)
   ↪  error
21
22 type callbackMapper func(entity ddiffer.Entity, version string, read
   ↪  func(data interface{}) error, save func(id string, data interface{},
   ↪  key, value string) error) error
23
24 func getParentID(lastPeriod, lastUniversity string) string {
25         result := lastPeriod
26         if len(result) == 0 {
27                 result = lastUniversity
28         }
29         return result
30 }
31
```

```
32 func manageDataset(target ddiffer.TargetType, version string, decoder
   ↪ coder.StreamingDecoder, cb callback, process func(id string, data
   ↪ interface{}) error) error {
33         getter, err := ddiffer.GetGetter(target)
34 
35         if err == nil {
36                 entity := getter()
37                 for err == nil {
38                         entity.Clean()
39                         err = cb(entity, version, func(data interface{})
                           ↪ error {
40                                 return decoder.Decode(data)
41                         }, process)
42                 }
43                 entity.Put()
44         }
45         if errors.Cause(err) == io.EOF {
46                 err = nil
47         }
48         return err
49 }
50 
51 func importDataset(target ddiffer.TargetType, version string, decoder
   ↪ coder.StreamingDecoder, store kv.Store, table string, cb callback)
   ↪ error {
52         batch := store.Batch()
53         err := manageDataset(target, version, decoder, cb, func(id string,
           ↪ data interface{}) error {
54                 return batch.Set(table, id, data)
55         })
56         if err == nil {
57                 err = batch.Commit()
58         }
59         return err
60 }
61 
62 func deleteDataset(target ddiffer.TargetType, version string, decoder
   ↪ coder.StreamingDecoder, store kv.Store, table string, cb callback)
   ↪ error {
```

```
63          return manageDataset(target, version, decoder, cb, func(id string,
            ↪  data interface{}) error {
64                  return store.Del(table, id)
65          })
66  }
67
68  func manageIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, mapper callbackMapper, callback func(entity
    ↪  ddiffer.Entity, id string, data interface{}, key, value string)
    ↪  error) error {
69          getter, err := ddiffer.GetGetter(target)
70          if err == nil {
71                  entity := getter()
72                  for err == nil {
73                          entity.Clean()
74                          err = mapper(entity, version, func(data
                            ↪  interface{}) error {
75                                  return decoder.Decode(data)
76                          }, func(id string, data interface{}, key, value
                            ↪  string) error {
77                                  return callback(entity, id, data, key,
                                    ↪  value)
78                          })
79                  }
80                  entity.Put()
81          }
82          if errors.Cause(err) == io.EOF {
83                  err = nil
84          }
85          return err
86  }
87
88  func importIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error), mapper callbackMapper) error {
89          var batch indexer.Batch
90          var lastID string
91          var lastUniversity string
92          var lastPeriod string
```

```
93          err := manageIndex(target, version, decoder, mapper, func(entity
            ↪ ddiffer.Entity, id string, data interface{}, key, value
            ↪ string) error {
94                  var saveErr error
95                  if batch == nil || lastUniversity !=
                    ↪ entity.GetUniversityID() || lastPeriod !=
                    ↪ entity.GetPeriodID() {
96                          if batch != nil {
97                                  saveErr = batch.Commit()
98                          }
99                          lastPeriod = entity.GetPeriodID()
100                         lastUniversity = entity.GetUniversityID()
101                         var index indexer.Index
102                         if saveErr == nil {
103                                 index, saveErr = getIndex(IndexData{
104                                         Target:   target,
105                                         ParentID: getParentID(lastPeriod,
                                            ↪ lastUniversity),
106                                 })
107                         }
108                         if saveErr == nil {
109                                 batch, saveErr = index.Update()
110                         }
111                 }
112                 if id != lastID && saveErr == nil {
113                         // Only make sense to do this call if it's an
                            ↪ update to
114                         // a previous version
115                         saveErr = batch.Delete(id)
116                         lastID = id
117                 }
118                 if saveErr == nil {
119                         if key == "" || key == "default" {
120                                 saveErr = batch.Set(value, id, data)
121                         } else {
122                                 saveErr = batch.SetField(key, value, id,
                                    ↪ data)
123                         }
124                 }
```

```
125                 return saveErr
126         })
127         if err == nil && batch != nil {
128                 err = batch.Commit()
129         }
130         return err
131 }
132
133 func deleteIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error), mapper callbackMapper) error {
134         var batch indexer.Batch
135         var lastPeriod string
136         var lastUniversity string
137         var lastID string
138         err := manageIndex(target, version, decoder, mapper, func(entity
            ↪  ddiffer.Entity, id string, _ interface{}, _, _ string) error
            ↪  {
139                 var saveErr error
140                 if batch == nil || lastUniversity !=
                    ↪  entity.GetUniversityID() || lastPeriod !=
                    ↪  entity.GetPeriodID() {
141                         if batch != nil {
142                                 saveErr = batch.Commit()
143                         }
144                         lastUniversity = entity.GetUniversityID()
145                         lastPeriod = entity.GetPeriodID()
146                         var index indexer.Index
147                         if saveErr == nil {
148                                 index, saveErr = getIndex(IndexData{
149                                         Target:   target,
150                                         ParentID: getParentID(lastPeriod,
                                            ↪  lastUniversity),
151                                 })
152                         }
153                         if saveErr == nil {
154                                 batch, saveErr = index.Update()
155                         }
156                 }
```

```
157                 if id != lastID && saveErr == nil {
158                         saveErr = batch.Delete(id)
159                         lastID = id
160                 }
161                 return saveErr
162         })
163         if batch != nil && err == nil {
164                 err = batch.Commit()
165         }
166         return err
167 }
168
169 func optimizeIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error), mapper callbackMapper) error {
170         var lastUniversity string
171         var lastPeriod string
172         return manageIndex(target, version, decoder, mapper, func(entity
            ↪  ddiffer.Entity, id string, data interface{}, key, value
            ↪  string) error {
173                 var saveErr error
174                 if lastUniversity != entity.GetUniversityID() ||
                    ↪  lastPeriod != entity.GetPeriodID() {
175                         lastUniversity = entity.GetUniversityID()
176                         lastPeriod = entity.GetPeriodID()
177                         var index indexer.Index
178                         index, saveErr = getIndex(IndexData{
179                                 Target:   target,
180                                 ParentID: getParentID(lastPeriod,
                                    ↪  lastUniversity),
181                         })
182                         if saveErr == nil {
183                                 saveErr = index.Optimize()
184                         }
185                         if saveErr == nil {
186                                 saveErr = index.RunGC()
187                         }
188                 }
189                 return saveErr
```

```
190         })
191 }
```

Arquivo campus.go

```
1  package importer
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6  )
7
8  func importCampi(entity ddiffer.Entity, version string, read func(data
   ↪  interface{}) error, save func(id string, data interface{}) error)
   ↪  error {
9          row := entity.(*ddiffer.DiffCampus)
10         err := read(row)
11         if err == nil {
12                 result := models.Campus{
13                         ID:                row.ID,
14                         UniversityID:      row.UniversityID,
15                         Name:              row.Campus.Name,
16                         Version:           version,
17                         OLCode:            row.Campus.OLCode,
18                         Period:            toModelsForeignKey(row.Period),
19                         DisciplineOffers:
                           ↪  toModelsForeignKeys(row.Disciplines),
20                 }
21                 err = save(row.ID, &result)
22         }
23         return err
24 }
```

Arquivo course.go

```
1  package importer
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
```

```
5         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6 )
7
8 func importCourses(entity ddiffer.Entity, version string, read func(data
  ↪  interface{}) error, save func(id string, data interface{}) error)
  ↪  error {
9         row := entity.(*ddiffer.DiffCourse)
10        err := read(row)
11        result := models.Course{
12                ID:             row.ID,
13                UniversityID:   row.UniversityID,
14                Name:           row.Course.Name,
15                Version:        version,
16                URL:            row.Course.URL,
17                CourseVersion: row.Course.Version,
18                Habilitations: toModelsForeignKeys(row.Habilitations),
19        }
20        if err == nil {
21                err = save(row.ID, &result)
22        }
23        return err
24 }
```

Arquivo discipline.go

```
1 package importer
2
3 import (
4         "github.com/fjorgemota/gurudamatricula/models"
5         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6 )
7
8 func importDisciplines(entity ddiffer.Entity, version string, read
  ↪  func(data interface{}) error, save func(id string, data interface{})
  ↪  error) error {
9         row := entity.(*ddiffer.DiffDiscipline)
10        err := read(row)
11        result := models.Discipline{
12                ID:           row.ID,
13                GenericID:    row.GenericID,
```

```
14              UniversityID: row.UniversityID,
15              Habilitation: toModelsForeignKey(row.Habilitation),
16              Course:       toModelsForeignKey(row.Course),
17              Step:         toModelsForeignKey(row.Step),
18              Code:         row.Discipline.Code,
19              Name:         row.Discipline.Name,
20              Type:         row.Discipline.Type,
21              Description:  row.Discipline.Description,
22              Version:      version,
23      }
24      for i, related := range row.Discipline.Related {
25              result.Related = append(result.Related,
        ↪  models.DisciplineRelated{
26                      Name: related.Name,
27                      Type: related.Type,
28              })
29              for _, relatedItem := range related.Items {
30                      result.RelatedItems = append(result.RelatedItems,
                        ↪  models.DisciplineRelatedItem{
31                              RelatedID:   i,
32                              Type:        relatedItem.Type,
33                              Value:       relatedItem.Value,
34                              Description: relatedItem.Description,
35                      })
36              }
37      }
38      result.Workload = make([]models.DisciplineWorkload,
        ↪  len(row.Discipline.Workload))
39      for i, workload := range row.Discipline.Workload {
40              result.Workload[i].ID = workload.ID
41              result.Workload[i].Name = workload.Name
42              result.Workload[i].Unit = workload.Unit
43              result.Workload[i].Value = workload.Value
44      }
45      result.Tables = make([]models.Table, len(row.Discipline.Tables))
46      for i, table := range row.Discipline.Tables {
47              result.Tables[i] = models.Table{
48                      ID:          table.ID,
49                      Title:       table.Title,
```

```
50                          Description: table.Description,
51                  }
52                  var start int
53                  result.TableColumns, start = convertColumns(i,
                    ↪  table.Columns, result.TableColumns)
54                  result.TableRows = convertRows(i,
                    ↪  result.TableColumns[start:], start, table.Rows,
                    ↪  result.TableRows)
55          }
56          if err == nil {
57                  err = save(row.ID, &result)
58          }
59          return err
60  }
```

Arquivo discipline_index.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6   )
7
8   func convertDiscipline(entity *ddiffer.DiffDiscipline, version string)
    ↪  (models.BasicDiscipline, error) {
9           var err error
10          result := models.BasicDiscipline{
11                  ID:           entity.ID,
12                  GenericID:    entity.GenericID,
13                  Habilitation: toModelsForeignKey(entity.Habilitation),
14                  Course:       toModelsForeignKey(entity.Course),
15                  Step:         toModelsForeignKey(entity.Step),
16                  Code:         entity.Discipline.Code,
17                  Name:         entity.Discipline.Name,
18                  Type:         entity.Discipline.Type,
19                  Description:  entity.Discipline.Description,
20                  Version:      version,
21          }
22          for i, related := range entity.Discipline.Related {
```

```
23                  result.Related = append(result.Related,
                    ↪  models.DisciplineRelated{
24                          Name: related.Name,
25                          Type: related.Type,
26                  })
27                  for _, relatedItem := range related.Items {
28                          result.RelatedItems = append(result.RelatedItems,
                            ↪  models.DisciplineRelatedItem{
29                                  RelatedID:   i,
30                                  Type:        relatedItem.Type,
31                                  Value:       relatedItem.Value,
32                                  Description: relatedItem.Description,
33                          })
34                  }
35          }
36          return result, err
37  }
38
39  func importDisciplinesIndex(entity ddiffer.Entity, version string, read
    ↪  func(data interface{}) error, save func(id string, data interface{},
    ↪  key, value string) error) error {
40          row := entity.(*ddiffer.DiffDiscipline)
41          err := read(row)
42          var result models.BasicDiscipline
43          if err == nil {
44                  result, err = convertDiscipline(row, version)
45          }
46          if err == nil {
47                  err = save(row.ID, &result, "default",
                    ↪  row.Discipline.Code)
48          }
49          if err == nil {
50                  err = save(row.ID, &result, "code", row.Discipline.Code)
51          }
52          if err == nil {
53                  err = save(row.ID, &result, "default",
                    ↪  row.Discipline.Name)
54          }
55          if err == nil {
```

```
56                  err = save(row.ID, &result, "name", row.Discipline.Name)
57          }
58          if err == nil {
59                  err = save(row.ID, &result, "generic_id", row.GenericID)
60          }
61          if err == nil {
62                  err = save(row.ID, &result, "course_id",
                    ↪  result.Course.ID)
63          }
64          if err == nil {
65                  err = save(row.ID, &result, "id", row.ID)
66          }
67          if err == nil {
68                  err = save(row.ID, &result, "habilitation_id",
                    ↪  result.Habilitation.ID)
69          }
70          if err == nil {
71                  err = save(row.ID, &result, "type", row.Discipline.Type)
72          }
73          if err == nil {
74                  mapping := map[string]string{
75                          "prerequisite": "no",
76                          "equivalent":   "no",
77                          "set":          "no",
78                  }
79                  for _, related := range result.RelatedItems {
80                          mapping[related.Type] = "yes"
81                  }
82                  for key, value := range mapping {
83                          if err == nil {
84                                  err = save(row.ID, &result,
                                    ↪  "has_"+key+"s", value)
85                          }
86                  }
87          }
88          for _, related := range result.RelatedItems {
89                  key := "related_"
90                  switch result.Related[related.RelatedID].Type {
91                  case "prerequisite":
```

```
92                      key += "depends_on"
93              case "equivalent":
94                      key += "equivalent_to"
95              case "set":
96                      key += "set_with"
97              }
98              if err == nil && key != "" {
99                      err = save(row.ID, &result, key, related.Value)
100             }
101             if err == nil && key != "" {
102                     err = save(row.ID, &result, key+"_description",
                        ↪  related.Description)
103             }
104     }
105     return err
106 }
```

Arquivo discipline_offer.go

```
1  package importer
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6          "github.com/fjorgemota/gurudamatricula/robot/format"
7  )
8
9  func importDisciplineOffers(entity ddiffer.Entity, version string, read
   ↪  func(data interface{}) error, save func(id string, data interface{})
   ↪  error) error {
10         row := entity.(*ddiffer.DiffDisciplineOffer)
11         err := read(row)
12         result := models.DisciplineOffer{
13                 ID:           row.ID,
14                 GenericID:    row.GenericID,
15                 Period:       toModelsForeignKey(row.Period),
16                 Campus:       toModelsForeignKey(row.Campus),
17                 UniversityID: row.UniversityID,
18                 Code:         row.DisciplineOffer.Code,
19                 Name:         row.DisciplineOffer.Name,
```

```
20                  Version:      version,
21          }
22          indexes := make(map[string]int)
23          schedules := make(map[format.Schedule]int)
24          for i, team := range row.Teams {
25                  if err == nil {
26                          result.Teams = append(result.Teams,
                            ↪ models.DisciplineOfferTeam{
27                                  Code: team.Code,
28                                  ID:   team.ID,
29                                  Course: models.DisciplineOfferTeamCourse{
30                                          Exclusivity:
                                            ↪ string(team.Course.Exclusive),
31                                          ID:          team.Course.ID,
32                                          Name:        team.Course.Name,
33                                  },
34                                  Vacancies:
                                    ↪ models.DisciplineOfferTeamVacancy{
35                                          Filled:  team.Vacancies.Filled,
36                                          Offered: team.Vacancies.Offered,
37                                  },
38                          })
39                  }
40                  for _, teacher := range team.Teachers {
41                          if _, ok := indexes[teacher.ID]; !ok {
42                                  indexes[teacher.ID] =
                                    ↪ len(result.Teachers)
43                                  result.Teachers = append(result.Teachers,
                                    ↪ models.DisciplineOfferTeacher{
44                                          ID:        teacher.ID,
45                                          GenericID:
                                            ↪ row.GetGenericID(teacher.ID),
46                                          Name:      teacher.Name,
47                                  })
48                          }
49                          if err == nil {
50                                  result.TeamTeachers =
                                    ↪ append(result.TeamTeachers,
                                    ↪ models.DisciplineOfferTeamTeacher{
```

```
51                                      TeacherID: indexes[teacher.ID],
52                                      TeamID:    i,
53                              })
54                      }
55              }
56              for _, schedule := range team.Schedules {
57                      roomIndex := -1
58                      if err == nil && len(schedule.Room.ID) > 0 {
59                              if _, ok := indexes[schedule.Room.ID];
                                ↪  !ok {
60                                      var room
                                        ↪  models.DisciplineOfferRoom
61                                      room.ID = schedule.Room.ID
62                                      room.GenericID =
                                        ↪  row.GetGenericID(schedule.Room.ID)
63                                      room.Name = schedule.Room.Name
64                                      room.OLCode =
                                        ↪  schedule.Room.OLCode
65                                      room.Description =
                                        ↪  schedule.Room.Description
66                                      indexes[room.ID] =
                                        ↪  len(result.Rooms)
67                                      result.Rooms =
                                        ↪  append(result.Rooms, room)
68                              }
69                              roomIndex = indexes[schedule.Room.ID]
70                      }
71                      if _, ok := schedules[schedule]; !ok {
72                              schedules[schedule] =
                                ↪  len(result.Schedules)
73                              result.Schedules =
                                ↪  append(result.Schedules,
                                ↪  models.DisciplineOfferSchedule{
74                                      DayOfWeek: schedule.DayOfWeek,
75                                      Start:
                                        ↪  models.DisciplineOfferTeamScheduleHourMinute
76                                              Hour:
                                                ↪  schedule.Start.Hour,
```

```
77                                              Minute:
                                                ↪ schedule.Start.Minute,
78                                      },
79                                      End:
                                        ↪ models.DisciplineOfferTeamScheduleHourM
80                                              Hour:
                                                ↪ schedule.End.Hour,
81                                              Minute:
                                                ↪ schedule.End.Minute,
82                                      },
83                              })
84                      }
85                      result.TeamSchedules =
                        ↪ append(result.TeamSchedules,
                        ↪ models.DisciplineOfferTeamSchedule{
86                              RoomID:     roomIndex,
87                              ScheduleID: schedules[schedule],
88                              TeamID:     i,
89                      })
90              }
91              for _, table := range team.Tables {
92                      idTable := len(result.TeamTables)
93                      result.TeamTables = append(result.TeamTables,
                        ↪ models.DisciplineOfferTeamTable{
94                              TeamID:      i,
95                              ID:          table.ID,
96                              Title:       table.Title,
97                              Description: table.Description,
98                      })
99                      var start int
100                     result.TableColumns, start =
                        ↪ convertColumns(idTable, table.Columns,
                        ↪ result.TableColumns)
101                     result.TableRows = convertRows(idTable,
                        ↪ result.TableColumns[start:], start,
                        ↪ table.Rows, result.TableRows)
102             }
103     }
104     if err == nil {
```

```
105                 err = save(row.ID, &result)
106         }
107         return err
108 }
```

Arquivo discipline_offer_index.go

```
1  package importer
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6  )
7
8  func convertDisciplineOffer(entity *ddiffer.DiffDisciplineOffer, version
   ↪  string) (models.DisciplineOffer, error) {
9          var err error
10         result := models.DisciplineOffer{
11                 ID:           entity.ID,
12                 GenericID:    entity.GenericID,
13                 UniversityID: entity.UniversityID,
14                 Period:       toModelsForeignKey(entity.Period),
15                 Campus:       toModelsForeignKey(entity.Campus),
16                 Code:         entity.DisciplineOffer.Code,
17                 Name:         entity.DisciplineOffer.Name,
18                 Version:      version,
19         }
20         return result, err
21 }
22
23 func importDisciplineOffersIndex(entity ddiffer.Entity, version string,
   ↪  read func(data interface{}) error, save func(id string, data
   ↪  interface{}, key, value string) error) error {
24         row := entity.(*ddiffer.DiffDisciplineOffer)
25         err := read(row)
26         var result models.DisciplineOffer
27         if err == nil {
28                 result, err = convertDisciplineOffer(row, version)
29         }
30         if err == nil {
```

```
31                  err = save(row.ID, &result, "id", row.ID)
32          }
33          if err == nil {
34                  err = save(row.ID, &result, "generic_id",
                    ↪  result.GenericID)
35          }
36          if err == nil {
37                  err = save(row.ID, &result, "campus_id",
                    ↪  result.Campus.ID)
38          }
39          if err == nil {
40                  err = save(row.ID, &result, "name", result.Name)
41          }
42          if err == nil {
43                  err = save(row.ID, &result, "default", result.Name)
44          }
45          if err == nil {
46                  err = save(row.ID, &result, "code", result.Code)
47          }
48          if err == nil {
49                  err = save(row.ID, &result, "default", result.Code)
50          }
51          return err
52  }
```

Arquivo foreign_key.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6   )
7
8   func toModelsForeignKeys(items []ddiffer.DiffForeignKey)
    ↪  []models.ForeignKey {
9           result := make([]models.ForeignKey, len(items))
10          for i, item := range items {
11                  result[i] = toModelsForeignKey(item)
12          }
```

```
13          return result
14  }
15
16  func toModelsForeignKey(item ddiffer.DiffForeignKey) models.ForeignKey {
17          return models.ForeignKey{
18                  ID:   item.ID,
19                  Name: item.Name,
20          }
21  }
```

Arquivo habilitation.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6   )
7
8   func importHabilitations(entity ddiffer.Entity, version string, read
    ↪  func(data interface{}) error, save func(id string, data interface{})
    ↪  error) error {
9           row := entity.(*ddiffer.DiffHabilitation)
10          err := read(row)
11          if err == nil {
12                  result := models.Habilitation{
13                          ID:           row.ID,
14                          UniversityID: row.UniversityID,
15                          Name:         row.Habilitation.Name,
16                          Version:      version,
17                          Course:       toModelsForeignKey(row.Course),
18                  }
19                  for i, limit := range row.Habilitation.Limits {
20                          result.Limits = append(result.Limits,
                            ↪  models.HabilitationLimit{
21                                  Context: limit.Context,
22                                  ID:      limit.ID,
23                                  Name:    limit.Name,
24                                  Unit:    limit.Unit,
25                          })
```

```
26                      for key, value := range limit.Values {
27                              result.LimitValues =
                                ↪  append(result.LimitValues,
                                ↪  models.HabilitationLimitValue{
28                                      ID:    i,
29                                      Key:   key,
30                                      Value: value,
31                              })
32                      }
33              }
34              for i, step := range row.Habilitation.Steps {
35                      result.Steps = append(result.Steps,
                        ↪  models.HabilitationStep{
36                              ID:   step.ID,
37                              Name: step.Name,
38                      })
39                      for _, discipline := range step.Disciplines {
40                              result.StepDisciplines =
                                ↪  append(result.StepDisciplines,
                                ↪  models.HabilitationStepDiscipline{
41                                      StepID: i,
42                                      Discipline: models.ForeignKey{
43                                              ID:   discipline.ID,
44                                              Name: discipline.Name,
45                                      },
46                              })
47                      }
48              }
49              result.Tables = make([]models.Table,
                ↪  len(row.Habilitation.Tables))
50              for i, table := range row.Habilitation.Tables {
51                      result.Tables[i] = models.Table{
52                              ID:          table.ID,
53                              Title:       table.Title,
54                              Description: table.Description,
55                      }
56                      var start int
57                      result.TableColumns, start = convertColumns(i,
                        ↪  table.Columns, result.TableColumns)
```

```
58                          result.TableRows = convertRows(i,
                            ↪  result.TableColumns[start:], start,
                            ↪  table.Rows, result.TableRows)
59                  }
60                  err = save(row.ID, &result)
61          }
62          return err
63  }
```

Arquivo import.go

```
1   package importer
2
3   import (
4           "errors"
5
6           "github.com/fjorgemota/gurudamatricula/models"
7           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
8           "github.com/fjorgemota/gurudamatricula/util/coder"
9           "github.com/fjorgemota/gurudamatricula/util/indexer"
10          "github.com/fjorgemota/gurudamatricula/util/kv"
11  )
12
13  var errImporterNotFound = errors.New("importer: Importer not found")
14
15  type importerCallback struct {
16          importer callback
17          table    string
18  }
19
20  var importerTypes = map[ddiffer.TargetType]importerCallback{
21          ddiffer.TargetCampus: {
22                  importer: importCampi,
23                  table:    models.TableCampus,
24          },
25          ddiffer.TargetDisciplineOffer: {
26                  importer: importDisciplineOffers,
27                  table:    models.TableDisciplineOffers,
28          },
29          ddiffer.TargetPeriod: {
```

```
30                  importer: importPeriods,
31                  table:    models.TablePeriod,
32          },
33          ddiffer.TargetCourse: {
34                  importer: importCourses,
35                  table:    models.TableCourse,
36          },
37          ddiffer.TargetDiscipline: {
38                  importer: importDisciplines,
39                  table:    models.TableDiscipline,
40          },
41          ddiffer.TargetHabilitation: {
42                  importer: importHabilitations,
43                  table:    models.TableHabilitations,
44          },
45  }
46
47  var indexerTypes = map[ddiffer.TargetType]callbackMapper{
48          ddiffer.TargetDiscipline:      importDisciplinesIndex,
49          ddiffer.TargetDisciplineOffer: importDisciplineOffersIndex,
50          ddiffer.TargetRoom:            importRoomsIndex,
51          ddiffer.TargetTeam:            importTeamsIndex,
52          ddiffer.TargetTeacher:         importTeachersIndex,
53  }
54
55  func HasDataset(target ddiffer.TargetType) bool {
56          _, ok := importerTypes[target]
57          return ok
58  }
59
60  func HasIndex(target ddiffer.TargetType) bool {
61          _, ok := indexerTypes[target]
62          return ok
63  }
64
65  func ImportDataset(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, store kv.Store) error {
66          err := errImporterNotFound
67          if HasDataset(target) {
```

```
68                  result := importerTypes[target]
69                  err = importDataset(target, version, decoder, store,
                    ↪  result.table, result.importer)
70          }
71          return err
72  }
73
74  func DeleteDataset(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, store kv.Store) error {
75          err := errImporterNotFound
76          if HasDataset(target) {
77                  result := importerTypes[target]
78                  err = deleteDataset(target, version, decoder, store,
                    ↪  result.table, result.importer)
79          }
80          return err
81  }
82
83  func ImportIntoIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error)) error {
84          err := errImporterNotFound
85          if HasIndex(target) {
86                  mapper := indexerTypes[target]
87                  err = importIndex(target, version, decoder, getIndex,
                    ↪  mapper)
88          }
89          return err
90  }
91
92  func OptimizeIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error)) error {
93          err := errImporterNotFound
94          if HasIndex(target) {
95                  mapper := indexerTypes[target]
96                  err = optimizeIndex(target, version, decoder, getIndex,
                    ↪  mapper)
97          }
```

```
98          return err
99  }
100
101 func DeleteFromIndex(target ddiffer.TargetType, version string, decoder
    ↪  coder.StreamingDecoder, getIndex func(data IndexData) (indexer.Index,
    ↪  error)) error {
102         err := errImporterNotFound
103         if HasIndex(target) {
104                 mapper := indexerTypes[target]
105                 err = deleteIndex(target, version, decoder, getIndex,
                    ↪  mapper)
106         }
107         return err
108 }
```

Arquivo init.go

```
1   package importer
2
3   import (
4           "fmt"
5           "strconv"
6   )
7
8   var dayOfWeeks [7]string
9   var hours [24]string
10  var hourMinutes [24][60]string
11  var dayOfWeeksHourMinutes [7][24][60]string
12
13  func init() {
14          for dayOfWeek := range dayOfWeeks {
15                  dayOfWeeks[dayOfWeek] = strconv.Itoa(dayOfWeek + 1)
16          }
17          for hour := range hours {
18                  hours[hour] = fmt.Sprintf("%02d", hour)
19          }
20          var minutes [60]string
21          for minute := range minutes {
22                  minutes[minute] = fmt.Sprintf("%02d", minute)
23          }
```

```
24          for hour, hourString := range hours {
25                  for minute, minuteString := range minutes {
26                          hourMinutes[hour][minute] = hourString + ":" +
                            ↪  minuteString
27                  }
28          }
29          for dayOfWeek := range dayOfWeeksHourMinutes {
30                  for hour, hourString := range hours {
31                          for minute, minuteString := range minutes {
32                                  dayOfWeeksHourMinutes[dayOfWeek][hour][minute]
                                    ↪  = dayOfWeeks[dayOfWeek] + "-" +
                                    ↪  hourString + ":" + minuteString
33                          }
34                  }
35          }
36  }
```

Arquivo period.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6   )
7
8   func importPeriods(entity ddiffer.Entity, version string, read func(data
    ↪  interface{}) error, save func(id string, data interface{}) error)
    ↪  error {
9           row := entity.(*ddiffer.DiffPeriod)
10          err := read(row)
11          if err == nil {
12                  result := models.Period{
13                          ID:           row.ID,
14                          UniversityID: row.UniversityID,
15                          Name:         row.Period.Name,
16                          Campi:        make([]models.PeriodCampus,
                            ↪  len(row.Campi)),
17                          Version:      version,
18                  }
```

```
19                  for i, campus := range row.Campi {
20                          result.Campi[i] = models.PeriodCampus{
21                                  ID:     campus.ID,
22                                  Name:   campus.Name,
23                                  OLCode: campus.OLCode,
24                          }
25                  }
26                  err = save(row.ID, &result)
27          }
28          return err
29  }
```

Arquivo pipeline.go

```
1   package importer
2
3   import (
4           "context"
5           "io"
6
7           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
8           "github.com/fjorgemota/gurudamatricula/util/coder"
9           "github.com/fjorgemota/gurudamatricula/util/file"
10          "github.com/fjorgemota/gurudamatricula/util/indexer"
11          "github.com/fjorgemota/gurudamatricula/util/kv"
12          "github.com/fjorgemota/gurudamatricula/util/pipeline"
13  )
14
15  type PipelineDependencies interface {
16          ddiffer.Handler
17          GetManager(ctx context.Context) (file.Manager, error)
18          GetStore(ctx context.Context) (kv.Store, error)
19          GetIndex(ctx context.Context, data IndexData) (indexer.Index,
            ↪  error)
20  }
21
22  type PipelineTaskNameOptions struct {
23          ImportDataset        string
24          ImportChangedDataset string
25          DeleteDataset        string
```

```
26          ImportIndex          string
27          OptimizeIndex        string
28  }
29
30  type PipelineOptions struct {
31          TaskNames PipelineTaskNameOptions
32          Parallel  bool
33  }
34
35  type pipelineHandler struct {
36          dependencies PipelineDependencies
37          options      PipelineOptions
38  }
39
40  func (ph pipelineHandler) importChangedDataset(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, target ddiffer.TargetType, result
    ↪  string) error {
41          var err error
42          var store kv.Store
43          var reader io.ReadCloser
44          var decoder coder.StreamingDecoder
45          store, err = ph.dependencies.GetStore(ctx)
46          if err == nil {
47                  reader, err = ph.getReader(ctx, result)
48          }
49          if err == nil {
50                  decoder = ph.dependencies.GetStreamingDecoder(reader)
51          }
52          if err == nil {
53                  err = ImportDataset(target, version, decoder, store)
54          }
55          if err == nil {
56                  err = reader.Close()
57          }
58          return err
59  }
60
61  func (ph pipelineHandler) getReader(ctx context.Context, filename string)
    ↪  (io.ReadCloser, error) {
```

```
62          var reader io.ReadCloser
63          manager, err := ph.dependencies.GetManager(ctx)
64          if err == nil {
65                  reader, err = manager.Reader(filename)
66          }
67          return reader, err
68  }
69
70  func (ph pipelineHandler) deleteDataset(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, target ddiffer.TargetType, result
    ↪  string) error {
71          store, err := ph.dependencies.GetStore(ctx)
72          var reader io.ReadCloser
73          if err == nil {
74                  reader, err = ph.getReader(ctx, result)
75          }
76          var decoder coder.StreamingDecoder
77          if err == nil {
78                  decoder = ph.dependencies.GetStreamingDecoder(reader)
79          }
80          if err == nil {
81                  err = DeleteDataset(target, version, decoder, store)
82          }
83          if err == nil {
84                  err = reader.Close()
85          }
86          return err
87  }
88
89  func (ph pipelineHandler) importSequentialDataset(ctx context.Context,
    ↪  pipe pipeline.Pipeline, version string, result
    ↪  ddiffer.ComparerResult) error {
90          store, err := ph.dependencies.GetStore(ctx)
91          var toDelete bool
92          var filename string
93          newResult := result
94          if len(result.Deleted) > 0 {
95                  filename = result.Deleted[0]
96                  newResult.Deleted = result.Deleted[1:]
```

```
97                  toDelete = true
98          } else if len(result.Created) > 0 {
99                  filename = result.Created[0]
100                 newResult.Created = result.Created[1:]
101         } else if len(result.Modified) > 0 {
102                 filename = result.Modified[0]
103                 newResult.Modified = result.Modified[1:]
104         }
105         shouldContinue := !ddiffer.IsComparerResultEmpty(newResult)
106         reader, err := ph.getReader(ctx, filename)
107         var decoder coder.StreamingDecoder
108         if err == nil {
109                 decoder = ph.dependencies.GetStreamingDecoder(reader)
110         }
111         if err == nil {
112                 if toDelete {
113                         err = DeleteDataset(result.Target, version,
                        ↪  decoder, store)
114                 } else {
115                         err = ImportDataset(result.Target, version,
                        ↪  decoder, store)
116                 }
117         }
118         if err == nil {
119                 err = reader.Close()
120         }
121         if err == nil && shouldContinue {
122                 _, err = pipe.Dispatch(ph.options.TaskNames.ImportDataset,
                ↪  version, newResult)
123         }
124         return err
125 }
126
127 func (ph pipelineHandler) importParallelDataset(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, result ddiffer.ComparerResult)
    ↪  error {
128         var err error
129         for _, created := range result.Created {
130                 if err == nil {
```

```
131                         _, err =
                            ↪  pipe.Dispatch(ph.options.TaskNames.ImportChangedDataset
                            ↪  version, result.Target, created)
132                 }
133         }
134         for _, modified := range result.Modified {
135                 if err == nil {
136                         _, err =
                            ↪  pipe.Dispatch(ph.options.TaskNames.ImportChangedDataset
                            ↪  version, result.Target, modified)
137                 }
138         }
139         for _, deleted := range result.Deleted {
140                 if err == nil {
141                         _, err =
                            ↪  pipe.Dispatch(ph.options.TaskNames.DeleteDataset,
                            ↪  version, result.Target, deleted)
142                 }
143         }
144         return err
145 }
146 func (ph pipelineHandler) importDataset(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, result ddiffer.ComparerResult)
    ↪  (ddiffer.ComparerResult, error) {
147         var err error
148         if ph.options.Parallel {
149                 err = ph.importParallelDataset(ctx, pipe, version,
                    ↪  result)
150         } else {
151                 err = ph.importSequentialDataset(ctx, pipe, version,
                    ↪  result)
152         }
153         return result, err
154 }
155
156 func (ph pipelineHandler) importIndex(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, result ddiffer.ComparerResult)
    ↪  (ddiffer.ComparerResult, error) {
157         var err error
```

```
158         getIndex := func(data IndexData) (indexer.Index, error) {
159                 return ph.dependencies.GetIndex(ctx, data)
160         }
161         // We only need to know if it's to delete..
162         // If it's not, it needs to be to update or create, which is the
            ↪  same for us
163         var toDelete bool
164         var filename string
165         newResult := result
166         if len(result.Deleted) > 0 {
167                 filename = result.Deleted[0]
168                 newResult.Deleted = result.Deleted[1:]
169                 toDelete = true
170         } else if len(result.Created) > 0 {
171                 filename = result.Created[0]
172                 newResult.Created = result.Created[1:]
173         } else if len(result.Modified) > 0 {
174                 filename = result.Modified[0]
175                 newResult.Modified = result.Modified[1:]
176         }
177         shouldContinue := !ddiffer.IsComparerResultEmpty(newResult)
178         reader, err := ph.getReader(ctx, filename)
179         var decoder coder.StreamingDecoder
180         if err == nil {
181                 decoder = ph.dependencies.GetStreamingDecoder(reader)
182         }
183         if err == nil {
184                 // We always do incremental updates here because we are
                    ↪  reading
185                 // from partial files with only what changed, so if we
                    ↪  rebuild
186                 // the index here probably we will delete everything that
                    ↪  don't
187                 // changed
188                 if toDelete {
189                         err = DeleteFromIndex(result.Target, version,
                            ↪  decoder, getIndex)
190                 } else {
```

```
191                         err = ImportIntoIndex(result.Target, version,
                            ↪  decoder, getIndex)
192                 }
193         }
194         if err == nil {
195                 err = reader.Close()
196         }
197         if err == nil && shouldContinue {
198                 _, err = pipe.Dispatch(ph.options.TaskNames.ImportIndex,
                    ↪  version, newResult)
199         }
200         return result, err
201 }
202
203 func (ph pipelineHandler) optimizeIndex(ctx context.Context, pipe
    ↪  pipeline.Pipeline, version string, original ddiffer.ComparerResult,
    ↪  queue []string) (ddiffer.ComparerResult, error) {
204         var err error
205         getIndex := func(data IndexData) (indexer.Index, error) {
206                 return ph.dependencies.GetIndex(ctx, data)
207         }
208         var filename string
209         filename, queue = queue[0], queue[1:]
210         reader, err := ph.getReader(ctx, filename)
211         var decoder coder.StreamingDecoder
212         if err == nil {
213                 decoder = ph.dependencies.GetStreamingDecoder(reader)
214         }
215         if err == nil {
216                 err = OptimizeIndex(original.Target, version, decoder,
                    ↪  getIndex)
217         }
218         if err == nil {
219                 err = reader.Close()
220         }
221         if err == nil && len(queue) > 0 {
222                 _, err = pipe.Dispatch(ph.options.TaskNames.OptimizeIndex,
                    ↪  version, original, queue)
223         }
```

```
224         return original, err
225 }
226
227 func (ph pipelineHandler) ImportIndex(pipe pipeline.Pipeline, version
    ↪  string, input ddiffer.ComparerResult) (pipeline.FutureID, error) {
228         var result pipeline.FutureID
229         results, err := pipe.Dispatch(ph.options.TaskNames.ImportIndex,
            ↪  version, input)
230         if err == nil {
231                 result = results[0]
232         }
233         return result, err
234 }
235
236 func (ph pipelineHandler) OptimizeIndex(pipe pipeline.Pipeline, version
    ↪  string, result ddiffer.ComparerResult) (pipeline.FutureID, error) {
237         var queue []string
238         queue = append(queue, result.Created...)
239         queue = append(queue, result.Modified...)
240         queue = append(queue, result.Deleted...)
241         results, err := pipe.Dispatch(ph.options.TaskNames.OptimizeIndex,
            ↪  version, result, queue)
242         return results[0], err
243 }
244
245 func (ph pipelineHandler) ImportDataset(pipe pipeline.Pipeline, version
    ↪  string, input ddiffer.ComparerResult) (pipeline.FutureID, error) {
246         var result pipeline.FutureID
247         results, err := pipe.Dispatch(ph.options.TaskNames.ImportDataset,
            ↪  version, input)
248         if err == nil {
249                 result = results[0]
250         }
251         return result, err
252 }
253
254 func (ph pipelineHandler) Import(pipe pipeline.Pipeline, version string,
    ↪  input ddiffer.ComparerResult) (pipeline.FutureID, error) {
255         var result pipeline.FutureID
```

```
256         var futures []pipeline.FutureID
257         var err error
258         if err == nil && HasDataset(input.Target) {
259                 futures, err =
                    ↪ pipe.Dispatch(ph.options.TaskNames.ImportDataset,
                    ↪ version, input)
260                 if err == nil && HasIndex(input.Target) {
261                         if ph.options.Parallel {
262                                 futures, err =
                                    ↪ pipe.Dispatch(ph.options.TaskNames.ImportIndex,
                                    ↪ version, input)
263                         } else {
264                                 futures, err =
                                    ↪ pipe.Dispatch(ph.options.TaskNames.ImportIndex,
                                    ↪ version, futures[0])
265                         }
266                 }
267         } else if err == nil && HasIndex(input.Target) {
268                 futures, err =
                    ↪ pipe.Dispatch(ph.options.TaskNames.ImportIndex,
                    ↪ version, input)
269         }
270         if err == nil && len(futures) > 0 {
271                 result = futures[0]
272         }
273         return result, err
274 }
275
276 type PipelineHandler interface {
277         ImportIndex(pipe pipeline.Pipeline, version string, result
            ↪ ddiffer.ComparerResult) (pipeline.FutureID, error)
278         OptimizeIndex(pipe pipeline.Pipeline, version string, result
            ↪ ddiffer.ComparerResult) (pipeline.FutureID, error)
279         ImportDataset(pipe pipeline.Pipeline, version string, result
            ↪ ddiffer.ComparerResult) (pipeline.FutureID, error)
280         Import(pipe pipeline.Pipeline, version string, result
            ↪ ddiffer.ComparerResult) (pipeline.FutureID, error)
281 }
282
```

```
283 func getDefaultPipelineOptions(options PipelineOptions) PipelineOptions {
284         if options.TaskNames.DeleteDataset == "" {
285                 options.TaskNames.DeleteDataset =
                    ↪  "importer.internal.delete_dataset"
286         }
287         if options.TaskNames.ImportChangedDataset == "" {
288                 options.TaskNames.ImportChangedDataset =
                    ↪  "importer.internal.import_changed_dataset"
289         }
290         if options.TaskNames.ImportDataset == "" {
291                 options.TaskNames.ImportDataset =
                    ↪  "importer.import_dataset"
292         }
293         if options.TaskNames.ImportIndex == "" {
294                 options.TaskNames.ImportIndex = "importer.import_index"
295         }
296         if options.TaskNames.OptimizeIndex == "" {
297                 options.TaskNames.OptimizeIndex =
                    ↪  "importer.optimize_index"
298         }
299         return options
300 }
301
302 func NewPipelineHandler(dispatcher pipeline.Dispatcher, dependencies
    ↪  PipelineDependencies, options PipelineOptions) (PipelineHandler,
    ↪  error) {
303         options = getDefaultPipelineOptions(options)
304         handler := pipelineHandler{
305                 dependencies: dependencies,
306                 options:      options,
307         }
308         err := dispatcher.Register(options.TaskNames.DeleteDataset,
            ↪  handler.deleteDataset)
309         if err == nil {
310                 err =
                    ↪  dispatcher.Register(options.TaskNames.ImportChangedDataset,
                    ↪  handler.importChangedDataset)
311         }
312         if err == nil {
```

```
313                 err = dispatcher.Register(options.TaskNames.ImportDataset,
                    ↪  handler.importDataset)
314         }
315         if err == nil {
316                 err = dispatcher.Register(options.TaskNames.ImportIndex,
                    ↪  handler.importIndex)
317         }
318         if err == nil {
319                 err = dispatcher.Register(options.TaskNames.OptimizeIndex,
                    ↪  handler.optimizeIndex)
320         }
321         return handler, err
322 }
```

Arquivo room.go

```
1  package importer
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/models"
5          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6  )
7
8  func importRoomsIndex(entity ddiffer.Entity, version string, read
   ↪  func(data interface{}) error, save func(id string, data interface{},
   ↪  key, value string) error) error {
9          row := entity.(*ddiffer.DiffRoom)
10         err := read(row)
11         result := models.Room{
12                 ID:           row.ID,
13                 GenericID:    row.GenericID,
14                 UniversityID: row.UniversityID,
15                 Period:       toModelsForeignKey(row.Period),
16                 Campus:       toModelsForeignKey(row.Campus),
17                 Name:         row.Room.Name,
18                 Description:  row.Room.Description,
19                 OLCode:       row.Room.OLCode,
20         }
21         if err == nil {
22                 err = save(row.ID, &result, "id", row.ID)
```

```
23          }
24          if err == nil {
25                  err = save(row.ID, &result, "generic_id", row.GenericID)
26          }
27          if err == nil {
28                  err = save(row.ID, &result, "campus_id", row.Campus.ID)
29          }
30          if err == nil {
31                  err = save(row.ID, &result, "name", result.Name)
32          }
33          if err == nil {
34                  err = save(row.ID, &result, "default", result.Name)
35          }
36          if err == nil {
37                  err = save(row.ID, &result, "description",
                    ↪  result.Description)
38          }
39          if err == nil {
40                  err = save(row.ID, &result, "olcode", result.OLCode)
41          }
42          return err
43  }
```

Arquivo table.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/format"
6   )
7
8   func convertColumns(idTable int, columns []format.Column, target
    ↪  []models.TableColumn) ([]models.TableColumn, int) {
9           start := len(target)
10          for _, column := range columns {
11                  target = append(target, models.TableColumn{
12                          TableID: idTable,
13                          ID:      column.ID,
14                          Name:    column.Name,
```

```
15                  })
16          }
17          return target, start
18  }
19
20  func convertRows(idTable int, columns []models.TableColumn, start int,
    ↪  rows []map[string]string, target []models.TableRow) []models.TableRow
    ↪  {
21          for k, row := range rows {
22                  for l, column := range columns {
23                          target = append(target, models.TableRow{
24                                  Row:    k,
25                                  Column: start + l,
26                                  Value:  row[column.ID],
27                          })
28                  }
29          }
30          return target
31  }
```

Arquivo teacher.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
6   )
7
8   func importTeachersIndex(entity ddiffer.Entity, version string, read
    ↪  func(data interface{}) error, save func(id string, data interface{},
    ↪  key, value string) error) error {
9           row := entity.(*ddiffer.DiffTeacher)
10          err := read(row)
11          result := models.Teacher{
12                  ID:           row.ID,
13                  UniversityID: row.UniversityID,
14                  GenericID:    row.GenericID,
15                  Period:       toModelsForeignKey(row.Period),
16                  Version:      version,
```

```
17                  Name:          row.Teacher.Name,
18                  URL:           row.Teacher.URL,
19          }
20          if err == nil {
21                  err = save(row.ID, &result, "id", row.ID)
22          }
23          if err == nil {
24                  err = save(row.ID, &result, "generic_id", row.GenericID)
25          }
26          if err == nil {
27                  err = save(row.ID, &result, "name", result.Name)
28          }
29          if err == nil {
30                  err = save(row.ID, &result, "default", result.Name)
31          }
32          return err
33  }
```

Arquivo team_index.go

```
1   package importer
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/models"
5           "github.com/fjorgemota/gurudamatricula/models/keygen"
6           "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
7   )
8
9   func convertTeamDisciplineOffer(entity *ddiffer.DiffTeam)
    ↪  models.TeamDisciplineOffer {
10          discipline := entity.Team.Discipline
11          result := models.TeamDisciplineOffer{
12                  ID:        entity.DisciplineOfferID,
13                  GenericID: entity.DisciplineOfferGenericID,
14                  Code:      discipline.Code,
15                  Name:      discipline.Name,
16          }
17          return result
18  }
19
```

```
20 func convertTeamCourseOffer(entity *ddiffer.DiffTeam)
   ↪ models.TeamCourseOffer {
21         var result models.TeamCourseOffer
22         course := entity.Team.Course
23         if len(entity.CourseID) > 0 {
24                 result = models.TeamCourseOffer{
25                         ID:        entity.CourseID,
26                         Name:      course.Name,
27                         Exclusive: string(course.Exclusive),
28                 }
29         }
30         return result
31 }
32
33 func convertTeam(entity *ddiffer.DiffTeam, version string) models.Team {
34         return models.Team{
35                 ID:           entity.ID,
36                 UniversityID: entity.UniversityID,
37                 PeriodID:     entity.PeriodID,
38                 Code:         entity.Team.Code,
39                 Discipline:   convertTeamDisciplineOffer(entity),
40                 Course:       convertTeamCourseOffer(entity),
41                 Version:      version,
42         }
43 }
44
45 func importTeamsIndex(entity ddiffer.Entity, version string, read
   ↪ func(data interface{}) error, save func(id string, data interface{},
   ↪ key, value string) error) error {
46         row := entity.(*ddiffer.DiffTeam)
47         err := read(row)
48         var result models.Team
49         if err == nil {
50                 result = convertTeam(row, version)
51         }
52         var campusID string
53         if err == nil {
54                 campusID, err = keygen.GenerateCampusID(row.UniversityID,
                   ↪ row.Team.Period, row.Team.Campus)
```

```
55          }
56          if err == nil {
57                  err = save(row.ID, &result, "id", row.ID)
58          }
59          if err == nil {
60                  err = save(row.ID, &result, "campus_id", campusID)
61          }
62          if err == nil {
63                  err = save(row.ID, &result, "discipline_id",
                    ↪  result.Discipline.ID)
64          }
65          if err == nil {
66                  err = save(row.ID, &result, "discipline_generic_id",
                    ↪  result.Discipline.GenericID)
67          }
68          if err == nil {
69                  err = save(row.ID, &result, "discipline_code",
                    ↪  row.Team.Discipline.Code)
70          }
71          if err == nil {
72                  err = save(row.ID, &result, "discipline_name",
                    ↪  row.Team.Discipline.Name)
73          }
74          if err == nil {
75                  err = save(row.ID, &result, "default",
                    ↪  row.Team.Discipline.Code)
76          }
77          if err == nil {
78                  err = save(row.ID, &result, "default",
                    ↪  row.Team.Discipline.Name)
79          }
80          if err == nil {
81                  err = save(row.ID, &result, "code", row.Team.Code)
82          }
83          if err == nil {
84                  err = save(row.ID, &result, "course_id",
                    ↪  result.Course.ID)
85          }
86          if err == nil {
```

```
87                  full := "0"
88                  if row.Team.Vacancies.Filled >=
                    ↪  row.Team.Vacancies.Offered {
89                          full = "1"
90                  }
91                  err = save(row.ID, &result, "full", full)
92          }
93          for _, teacher := range row.Team.Teachers {
94                  var teacherID, teacherPeriodID string
95                  if err == nil {
96                          teacherID, err =
                            ↪  keygen.GenerateTeacherID(row.UniversityID,
                            ↪  teacher)
97                  }
98                  if err == nil {
99                          teacherPeriodID, err =
                            ↪  keygen.GenerateTeacherByPeriodID(row.UniversityID,
                            ↪  row.Team.Period, teacher)
100                 }
101                 if err == nil {
102                         err = save(row.ID, &result, "teacher_id",
                            ↪  teacherID)
103                 }
104                 if err == nil {
105                         err = save(row.ID, &result, "teacher_period_id",
                            ↪  teacherPeriodID)
106                 }
107         }
108         for _, schedule := range row.Team.Schedules {
109                 if err == nil && len(schedule.Room.ID) >= 0 {
110                         var roomID, roomPeriodID string
111                         roomID, err :=
                            ↪  keygen.GenerateRoomID(row.UniversityID,
                            ↪  row.Team.Campus, schedule.Room)
112                         if err == nil {
113                                 roomPeriodID, err =
                                    ↪  keygen.GenerateRoomByPeriodID(row.UniversityID,
                                    ↪  row.Team.Period, row.Team.Campus,
                                    ↪  schedule.Room)
```

```
114                         }
115                         if err == nil {
116                                 err = save(row.ID, &result, "room_id",
                                    ↪  roomID)
117                         }
118                         if err == nil {
119                                 err = save(row.ID, &result,
                                    ↪  "room_period_id", roomPeriodID)
120                         }
121                         if err == nil {
122                                 err = save(row.ID, &result, "room_olcode",
                                    ↪  schedule.Room.OLCode)
123                         }
124                 }
125                 dayOfWeek := dayOfWeeks[schedule.DayOfWeek-1]
126                 if err == nil {
127                         err = save(row.ID, &result, "schedule_day",
                            ↪  dayOfWeek)
128                 }
129                 hour := schedule.Start.Hour
130                 minute := schedule.Start.Minute
131                 if err == nil {
132                         err = save(row.ID, &result, "schedule_hour",
                            ↪  hours[hour])
133                 }
134                 for (hour < schedule.End.Hour || minute <
                    ↪  schedule.End.Minute) && err == nil && hour < 24 {
135                         if err == nil {
136                                 err = save(row.ID, &result,
                                    ↪  "schedule_time",
                                    ↪  hourMinutes[hour][minute])
137                         }
138                         if err == nil {
139                                 err = save(row.ID, &result,
                                    ↪  "schedule_day_time",
                                    ↪  dayOfWeeksHourMinutes[schedule.DayOfWeek-1][hour][mi
140                         }
141                         minute = (minute + 1) % 60
142                         if minute == 0 {
```

```
143                                     hour++
144                                     if err == nil {
145                                             err = save(row.ID, &result,
                                                ↪  "schedule_hour", hours[hour])
146                                     }
147                             }
148                     }
149
150             }
151             return err
152     }
```

B.1.14 Pasta robot/joiner

Arquivo base.go

```
1  package joiner
2
3  import (
4          "io"
5
6          "github.com/fjorgemota/gurudamatricula/util/coder"
7  )
8
9  // Handler have methods to get StreamingEncoder and StreamingDecoder
10 // based on io.Writer and io.Reader interfaces
11 type Handler interface {
12         GetStreamingEncoder(io.WriteCloser) coder.StreamingEncoder
13         GetStreamingDecoder(io.Reader) coder.StreamingDecoder
14 }
```

Arquivo habilitations.go

```
1  package joiner
2
3  import (
4          "github.com/fjorgemota/gurudamatricula/robot/format/pool"
5          "github.com/fjorgemota/gurudamatricula/util/coder"
6          "github.com/fjorgemota/gurudamatricula/util/file"
7  )
```

```
8
9  func joinHabilitations(decoder coder.StreamingDecoder, encoder
   ↪  coder.StreamingEncoder) error {
10         var err error
11         hab := pool.GetHabilitation()
12         for err == nil {
13                 err = decoder.Decode(hab)
14                 if err == nil {
15                         err = encoder.Encode(hab)
16                 }
17                 pool.CleanHabilitation(hab)
18         }
19         pool.PutHabilitation(hab)
20         return err
21 }
22
23 // JoinHabilitations will get a slice of files, a target filename, and
   ↪  will
24 // join all the data of the slice of files into the target filename
25 func JoinHabilitations(manager file.Manager, handler Handler, filenames
   ↪  []string, target string) error {
26         return join(manager, handler, filenames, target,
           ↪  joinHabilitations)
27 }
```

Arquivo joiner.go

```
1  package joiner
2
3  import (
4          "io"
5
6          "github.com/fjorgemota/gurudamatricula/util/coder"
7          "github.com/fjorgemota/gurudamatricula/util/file"
8          "github.com/pkg/errors"
9  )
10
11 func join(manager file.Manager, handler Handler, filenames []string,
   ↪  target string, join func(decoder coder.StreamingDecoder, encoder
   ↪  coder.StreamingEncoder) error) error {
```

```
12          writer, err := manager.Writer(target)
13          var encoder coder.StreamingEncoder
14          if err == nil {
15                  encoder = handler.GetStreamingEncoder(writer)
16          }
17          var toDelete []string
18          for _, filename := range filenames {
19                  var reader io.ReadCloser
20                  if err == nil {
21                          reader, err = manager.Reader(filename)
22                  }
23                  if err != nil && file.IsNotExistError(err) {
24                          err = nil
25                          continue
26                  }
27                  var decoder coder.StreamingDecoder
28                  if err == nil {
29                          decoder = handler.GetStreamingDecoder(reader)
30                          err = join(decoder, encoder)
31                  }
32                  if errors.Cause(err) == io.EOF {
33                          err = nil
34                  }
35                  if err == nil {
36                          toDelete = append(toDelete, filename)
37                  }
38                  if err == nil {
39                          err = encoder.Flush()
40                  }
41                  if err == nil {
42                          err = reader.Close()
43                  }
44          }
45          if err == nil {
46                  err = encoder.Close()
47          }
48          if err == nil {
49                  err = file.DeleteFiles(manager, toDelete)
50          }
```

```
51          return err
52  }
```

Arquivo teams.go

```
1   package joiner
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/robot/format/pool"
5           "github.com/fjorgemota/gurudamatricula/util/coder"
6           "github.com/fjorgemota/gurudamatricula/util/file"
7   )
8
9   func joinTeams(decoder coder.StreamingDecoder, encoder
    ↪  coder.StreamingEncoder) error {
10          var err error
11          team := pool.GetTeam()
12          for err == nil {
13                  err = decoder.Decode(team)
14                  if err == nil {
15                          err = encoder.Encode(team)
16                  }
17                  pool.CleanTeam(team)
18          }
19          pool.PutTeam(team)
20          return err
21  }
22
23  // JoinTeams will get a slice of files, a target filename, and will
24  // join all the data of the slice of files into the target filename
25  func JoinTeams(manager file.Manager, handler Handler, filenames []string,
    ↪  target string) error {
26          return join(manager, handler, filenames, target, joinTeams)
27  }
```

B.1.15 Pasta robot/publisher

Arquivo publisher.go

```
1  package publisher
2
3  import (
4          "context"
5          "errors"
6          "io"
7          "io/ioutil"
8          "net/http"
9          "net/url"
10         "strings"
11
12         "github.com/sirupsen/logrus"
13
14         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
15
16         uuid "github.com/gofrs/uuid"
17
18         "github.com/fjorgemota/gurudamatricula/util/file"
19         "github.com/fjorgemota/gurudamatricula/util/pipeline"
20 )
21
22 func UUIDFilenameGenerator() (string, error) {
23         var result string
24         id, err := uuid.NewV4()
25         if err == nil {
26                 result = id.String()
27         }
28         return result, err
29 }
30
31 type Tasks struct {
32         Start      string
33         Notify     string
34         DeleteFile string
35 }
36
37 type Options struct {
38         NotificationURL   string
39         UniversityID      string
```

```
40          Secret            string
41          SourceType        ddiffer.SourceType
42          Tasks             Tasks
43          FilenameGenerator func() (string, error)
44  }
45
46  type Handler interface {
47          GetHTTPClient(ctx context.Context, jar http.CookieJar)
            ↪  *http.Client
48          GetManager(ctx context.Context) (file.Manager, error)
49          GetLogger(ctx context.Context) logrus.FieldLogger
50  }
51
52  type Bot interface {
53          Start(pipe pipeline.Pipeline, target string) (pipeline.FutureID,
            ↪  error)
54  }
55
56  type botPublisher struct {
57          bot     Bot
58          options Options
59          handler Handler
60  }
61
62  func (bp botPublisher) deleteFile(ctx context.Context, pipe
    ↪  pipeline.Pipeline, filename string) error {
63          manager, err := bp.handler.GetManager(ctx)
64          if err == nil {
65                  err = manager.Delete(filename)
66          }
67          return err
68  }
69
70  func (bp botPublisher) start(ctx context.Context, pipe pipeline.Pipeline)
    ↪  error {
71          id, err := bp.options.FilenameGenerator()
72          var filename pipeline.FutureID
73          if err == nil {
74                  target := bp.options.UniversityID + "-" + id
```

```
75                  filename, err = bp.bot.Start(pipe, target)
76          }
77          if err == nil {
78                  _, err = pipe.Dispatch(bp.options.Tasks.Notify, filename)
79          }
80          return err
81  }
82
83  func (bp botPublisher) notify(ctx context.Context, pipe pipeline.Pipeline,
    ↪  filename string) error {
84          var response *http.Response
85          logger := bp.handler.GetLogger(ctx).WithField("university_id",
            ↪  bp.options.UniversityID)
86          logger.Info("Notifying system about update on university")
87          handle, err := url.Parse(bp.options.NotificationURL)
88          if err == nil {
89                  query := handle.Query()
90                  query.Set("university_id", bp.options.UniversityID)
91                  query.Set("secret", bp.options.Secret)
92                  query.Set("filename", filename)
93                  switch bp.options.SourceType {
94                  case ddiffer.SourceHabilitations:
95                          query.Set("source_type", "habilitations")
96                  case ddiffer.SourceOffers:
97                          query.Set("source_type", "offers")
98                  }
99                  handle.RawQuery = query.Encode()
100                 client := bp.handler.GetHTTPClient(ctx, nil)
101                 url := handle.String()
102                 logger.Infof("Sending request to '%s'", url)
103                 response, err = client.Get(url)
104         }
105         if err == nil && response != nil && response.Body != nil {
106                 _, err = io.Copy(ioutil.Discard, response.Body)
107                 if err == nil {
108                         err = response.Body.Close()
109                 }
110         }
111         logger.WithError(err).Info("System notified")
```

```
112         return err
113 }
114
115 func (bp botPublisher) Start(pipe pipeline.Pipeline) error {
116         err := errors.New("Cannot start task without bot registered")
117         if bp.bot != nil {
118                 _, err = pipe.Dispatch(bp.options.Tasks.Start)
119         }
120         return err
121 }
122
123 func (bp botPublisher) DeleteFile(pipe pipeline.Pipeline, filename
    ↪  string) error {
124         _, err := pipe.Dispatch(bp.options.Tasks.DeleteFile, filename)
125         return err
126 }
127
128 func getDefaultOptions(options Options) Options {
129         source := "." + strings.ToLower(options.SourceType.String())
130         if options.Tasks.Start == "" {
131                 options.Tasks.Start = options.UniversityID + source +
                    ↪  ".publisher.start"
132         }
133         if options.Tasks.Notify == "" {
134                 options.Tasks.Notify = options.UniversityID + source +
                    ↪  ".publisher.notify"
135         }
136         if options.Tasks.DeleteFile == "" {
137                 options.Tasks.DeleteFile = options.UniversityID + source
                    ↪  + ".publisher.delete_file"
138         }
139         if options.FilenameGenerator == nil {
140                 options.FilenameGenerator = UUIDFilenameGenerator
141         }
142         return options
143 }
144
145 type Publisher interface {
146         Start(pipe pipeline.Pipeline) error
```

```
147         DeleteFile(pipe pipeline.Pipeline, filename string) error
148 }
149
150 func NewBotPublisher(dispatcher pipeline.Dispatcher, bot Bot, handler
    ↪  Handler, options Options) (Publisher, error) {
151         options = getDefaultOptions(options)
152         publisher := botPublisher{
153                 bot:     bot,
154                 handler: handler,
155                 options: options,
156         }
157         err := dispatcher.Register(options.Tasks.DeleteFile,
          ↪  publisher.deleteFile)
158         if err == nil && bot != nil {
159                 err = dispatcher.Register(options.Tasks.Start,
                  ↪  publisher.start)
160                 if err == nil {
161                         err = dispatcher.Register(options.Tasks.Notify,
                          ↪  publisher.notify)
162                 }
163         }
164         return publisher, err
165 }
```

B.1.16 Pasta robot/ufsc2offers

Arquivo api.go

```
1  package ufsc2offers
2
3  import (
4          "fmt"
5          "math/rand"
6          "net/http"
7          "time"
8
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10         robotpool
           ↪  "github.com/fjorgemota/gurudamatricula/robot/format/pool"
11         "github.com/fjorgemota/gurudamatricula/util/clock"
```

```
12          "github.com/fjorgemota/gurudamatricula/util/coder"
13          "github.com/fjorgemota/gurudamatricula/util/kv"
14          "github.com/fjorgemota/gurudamatricula/util/pool"
15          "github.com/sirupsen/logrus"
16          "golang.org/x/net/html"
17  )
18
19  func randomSleep(clk clock.Clock, delay time.Duration) {
20          clk.Sleep(time.Duration((rand.Float64() + 0.5) * float64(delay)))
21  }
22
23  func Start(client *http.Client, dispatch func(bs []byte) error, cod
    ↪  coder.Coder, store kv.Store, clk clock.Clock, options Options) error
    ↪  {
24          ID, err := store.GenerateID(options.CheckTable)
25          var response bootResponse
26          if err == nil {
27                  randomSleep(clk, options.DelayBetweenRequests)
28                  response, err = bootRequest(client, store,
                    ↪  options.CheckTable, clk, options.NumberOfPeriods, ID,
                    ↪  options.UserAgent)
29          }
30          buf := pool.GetBytesBuffer()
31          for _, semester := range response.semesters {
32                  for _, campus := range response.campi {
33                          if err == nil {
34                                  buf.Reset()
35                                  err = cod.EncodeTo(&State{
36                                          Period: semester,
37                                          Campus: campus,
38                                          ID:     ID,
39                                          Prefix: fmt.Sprintf("%s-%s-%d",
                                            ↪  semester.Name, campus.Name,
                                            ↪  clk.Now().Unix()),
40                                  }, buf)
41                          }
42                          if err == nil {
43                                  err = dispatch(buf.Bytes())
44                          }
```

```
45                  }
46          }
47          return err
48  }
49
50  func FetchData(client *http.Client, logger logrus.FieldLogger, encoder
    ↪  coder.StreamingEncoder, store kv.Store, clk clock.Clock, options
    ↪  Options, actualState *State) (bool, error) {
51          var err error
52          if err == nil && actualState.ViewState == "" {
53                  var response bootResponse
54                  randomSleep(clk, options.DelayBetweenRequests)
55                  response, err = bootRequest(client, store,
                    ↪  options.CheckTable, clk, 0, actualState.ID,
                    ↪  options.UserAgent)
56                  actualState.ViewState = response.viewState
57          }
58          stringCache := pool.GetStringCache()
59          intCache := pool.GetAtoiCache()
60          courseCache := newCoursesCache(stringCache)
61          hasNext := true
62          var teams []*format.Team
63          for c := 0; c < options.BatchSize && hasNext && err == nil; c++ {
64                  var doc *html.Tokenizer
65                  if err == nil {
66                          randomSleep(clk, options.DelayBetweenRequests)
67                          actualState.PageNumber++
68                          doc, err = fetch(client, store,
                            ↪  options.CheckTable, clk, actualState.Period,
                            ↪  actualState.Campus, actualState.PageNumber,
                            ↪  actualState.ViewState, actualState.ID,
                            ↪  options.UserAgent)
69                  }
70                  if err == nil {
71                          teams = teams[:0]
72                          log := logger.WithField("page_number",
                            ↪  actualState.PageNumber)
```

```
73                      teams, hasNext, err = parse(teams, doc, log,
                        ↪  actualState.Period, actualState.Campus,
                        ↪  actualState.PageNumber, stringCache, intCache,
                        ↪  courseCache)
74              }
75              for _, team := range teams {
76                      if err == nil {
77                              err = encoder.Encode(team)
78                      }
79                      if err == nil {
80                              robotpool.PutTeam(team)
81                      }
82              }
83              if err == nil {
84                      err = encoder.Flush()
85              }
86      }
87      pool.PutStringCache(stringCache)
88      pool.PutAtoiCache(intCache)
89      return hasNext && err == nil, err
90  }
```

Arquivo bot.go

```
1   package ufsc2offers
2
3   import (
4           "bytes"
5           "fmt"
6           "io"
7           "net/http"
8
9           "github.com/fjorgemota/gurudamatricula/robot/joiner"
10          "github.com/fjorgemota/gurudamatricula/robot/util/ccookiejar"
11          "github.com/fjorgemota/gurudamatricula/util/clock"
12          "github.com/fjorgemota/gurudamatricula/util/coder"
13          "github.com/fjorgemota/gurudamatricula/util/file"
14          "github.com/fjorgemota/gurudamatricula/util/kv"
15          "github.com/fjorgemota/gurudamatricula/util/pipeline"
16          "github.com/fjorgemota/gurudamatricula/util/pool"
```

```
17          "github.com/sirupsen/logrus"
18          "golang.org/x/net/context"
19  )
20
21  // Handler defines a few constructors that Bot needs to work
22  type Handler interface {
23          GetStreamingEncoder(io.WriteCloser) coder.StreamingEncoder
24          GetHTTPClient(ctx context.Context, jar http.CookieJar)
            ↪  *http.Client
25          GetManager(context.Context) (file.Manager, error)
26          GetStore(context.Context) (kv.Store, error)
27          GetCookieJar() (ccookiejar.CodedCookieJar, error)
28          GetLogger(context.Context) logrus.FieldLogger
29  }
30
31  type bot struct {
32          cod     coder.Coder
33          options Options
34          pipe    pipeline.Pipeline
35          handler Handler
36          clk     clock.Clock
37  }
38
39  func (b bot) start(ctx context.Context, pipe pipeline.Pipeline, target
    ↪  string) (pipeline.FutureID, error) {
40          jar, err := b.handler.GetCookieJar()
41          var store kv.Store
42          if err == nil {
43                  store, err = b.handler.GetStore(ctx)
44          }
45          var files []interface{}
46          if err == nil {
47                  client := b.handler.GetHTTPClient(ctx, jar)
48                  var results []pipeline.FutureID
49                  err = Start(client, func(bs []byte) error {
50                          futIDs, errDispatch :=
                            ↪  pipe.DispatchWithOptions(b.options.Tasks.FetchCampus,
                            ↪  pipeline.TaskOptions{
51                                  MaxAttempts: 1,
```

```
52                          }, bs)
53                          if errDispatch == nil {
54                                  results = append(results, futIDs[0])
55                          }
56                          return errDispatch
57                  }, b.cod, store, b.clk, b.options)
58                  files = make([]interface{}, 0, len(results)+2)
59                  files = append(files, true)
60                  files = append(files, target)
61                  for _, result := range results {
62                          files = append(files, result)
63                  }
64          }
65          var result pipeline.FutureID
66          if err == nil {
67                  var results []pipeline.FutureID
68                  results, err = pipe.Dispatch(b.options.Tasks.JoinFiles,
                    ↪  files...)
69                  if err == nil {
70                          result = results[0]
71                  }
72          }
73          return result, err
74  }
75
76  func (b bot) Start(pipe pipeline.Pipeline, target string)
    ↪  (pipeline.FutureID, error) {
77          futIDs, err := pipe.DispatchWithOptions(b.options.Tasks.Start,
            ↪  pipeline.TaskOptions{
78                  MaxAttempts: 1,
79          }, target)
80          var futID pipeline.FutureID
81          if len(futIDs) > 0 {
82                  futID = futIDs[0]
83          }
84          return futID, err
85  }
86
87  func (b bot) generateName(actualState *State) string {
```

```
88          return fmt.Sprintf("%s-%d-%d-%d.partial_1", actualState.Prefix,
            ↪  actualState.PageNumber/b.options.BatchSize,
            ↪  b.clk.Now().Unix(), actualState.Retries)
89  }
90
91  func (b bot) joinFiles(ctx context.Context, pipe pipeline.Pipeline, final
    ↪  bool, target string, filenames ...string) (string, error) {
92          if len(filenames) == 1 && !final {
93                  // We just return the file we have, so we can avoid
                    ↪  writing a new file
94                  return filenames[0], nil
95          }
96          manager, err := b.handler.GetManager(ctx)
97          if err == nil {
98                  handle := handleJoiner{final: final, handle: b.handler}
99                  err = joiner.JoinTeams(manager, handle, filenames,
                    ↪  target)
100         }
101         return target, err
102 }
103
104 func (b bot) deleteFiles(ctx context.Context, pipe pipeline.Pipeline,
    ↪  filenames []string, bs []byte) (pipeline.FutureID, error) {
105         manager, err := b.handler.GetManager(ctx)
106         if err == nil {
107                 err = file.DeleteFiles(manager, filenames)
108         }
109         var futID pipeline.FutureID
110         if err == nil {
111                 var futIDs []pipeline.FutureID
112                 futIDs, err = pipe.Dispatch(b.options.Tasks.FetchCampus,
                    ↪  bs)
113                 if err == nil {
114                         futID = futIDs[0]
115                 }
116         }
117         return futID, err
118 }
119
```

```
120 func (b bot) fetchCampus(ctx context.Context, pipe pipeline.Pipeline, bs
    ↪  []byte) (pipeline.FutureID, error) {
121         id, _ := pipeline.GetID(ctx)
122         logger := b.handler.GetLogger(ctx).WithField("fut_id",
            ↪  id).WithField("package", "ufsc2")
123         actualState := &State{}
124         jar, err := b.handler.GetCookieJar()
125         var client *http.Client
126         if err == nil {
127                 client = b.handler.GetHTTPClient(ctx, jar)
128                 actualState.Cookies = jar
129         }
130         var store kv.Store
131         if err == nil {
132                 store, err = b.handler.GetStore(ctx)
133         }
134         var manager file.Manager
135         if err == nil {
136                 err = b.cod.DecodeFrom(bytes.NewReader(bs), actualState)
137         }
138         logger = logger.WithField("retries",
            ↪  actualState.Retries).WithField("campus",
            ↪  actualState.Campus.Name).WithField("semester",
            ↪  actualState.Period.Name)
139         logger.Debug("Attempting to fetch data")
140         name := b.generateName(actualState)
141         actualState.Names = append(actualState.Names, name)
142         if err == nil {
143                 manager, err = b.handler.GetManager(ctx)
144         }
145         var writer io.WriteCloser
146         if err == nil {
147                 writer, err = manager.Writer(name)
148         }
149         var hasNext bool
150         if err == nil {
151                 encoder := coder.NewGOBStreamingEncoder(writer)
152                 hasNext, err = FetchData(client, logger, encoder, store,
                    ↪  b.clk, b.options, actualState)
```

```
153                 if err == nil {
154                         err = encoder.Close()
155                 }
156         }
157         shouldDispatch := hasNext
158         var names []string
159         if err != nil {
160                 actualState.PageNumber = 0
161                 actualState.Retries++
162                 actualState.Cookies = nil
163                 actualState.ViewState = ""
164                 names = actualState.Names
165                 actualState.Names = nil
166                 shouldDispatch = actualState.Retries <
                    ↪ b.options.MaxRetries
167                 logger = logger.WithError(err).WithField("retries",
                    ↪ actualState.Retries).WithField("max_retries",
                    ↪ b.options.MaxRetries)
168                 if shouldDispatch {
169                         logger.Debug("Retrying again after error
                            ↪ detected")
170                         // Discard error because we will retry manually
171                         err = nil
172                 } else {
173                         logger.Debug("Aborting after error detected")
174                 }
175         }
176         var result pipeline.FutureID
177         var results []pipeline.FutureID
178         if shouldDispatch {
179                 buf := pool.GetBytesBuffer()
180                 defer pool.PutBytesBuffer(buf)
181                 err = b.cod.EncodeTo(actualState, buf)
182                 if err == nil {
183                         if names == nil {
184                                 results, err =
                                    ↪ pipe.Dispatch(b.options.Tasks.FetchCampus,
                                    ↪ buf.Bytes())
185                         } else {
```

```
186                                 results, err =
                                    ↪ pipe.Dispatch(b.options.Tasks.DeleteFiles,
                                    ↪ names, buf.Bytes())
187                         }
188                 }
189         } else if err == nil {
190                 files := make([]interface{}, 0, len(actualState.Names)+2)
191                 files = append(files, false)
192                 files = append(files, actualState.Prefix+".partial_2")
193                 for _, name := range actualState.Names {
194                         files = append(files, name)
195                 }
196                 logger.WithField("num_files",
                    ↪ len(files)-1).WithField("target",
                    ↪ files[0]).Debug("Dispatching task to join files")
197                 results, err = pipe.Dispatch(b.options.Tasks.JoinFiles,
                    ↪ files...)
198         }
199         if len(results) > 0 {
200                 result = results[0]
201         }
202         if err == nil {
203                 logger.Debug("Crawl task terminated successfully")
204         } else {
205                 logger.WithError(err).Error("Crawl Task terminated with
                    ↪ error")
206         }
207         return result, err
208 }
209
210 // Bot defines the basic interface to start the processing of the data
211 type Bot interface {
212         Start(pipe pipeline.Pipeline, target string) (pipeline.FutureID,
            ↪ error)
213 }
214
215 // NewBot Gets a new Bot
216 func NewBot(disp pipeline.Dispatcher, cod coder.Coder, clk clock.Clock,
    ↪ handler Handler, opt Options) (Bot, error) {
```

```
217         opt = getDefaults(opt)
218         instance := &bot{
219                 options: opt,
220                 cod:     cod,
221                 handler: handler,
222                 clk:     clk,
223         }
224         err := disp.Register(opt.Tasks.FetchCampus, instance.fetchCampus)
225         if err == nil {
226                 err = disp.Register(opt.Tasks.Start, instance.start)
227         }
228         if err == nil {
229                 err = disp.Register(opt.Tasks.DeleteFiles,
                    ↪  instance.deleteFiles)
230         }
231         if err == nil {
232                 err = disp.Register(opt.Tasks.JoinFiles,
                    ↪  instance.joinFiles)
233         }
234         return instance, err
235 }
```

Arquivo course.go

```
1  package ufsc2offers
2
3  import (
4          "bytes"
5          "errors"
6
7          "github.com/fjorgemota/gurudamatricula/util/pool"
8  )
9
10 var errCourseIdNotFound = errors.New("ufsc2: ID of the course not found")
11 var errCourseNameNotFound = errors.New("ufsc2: Name of the course not
   ↪  found")
12 var reserved = []byte("[Reservada Curso]")
13 var cancelled = []byte("[Cancelada]")
14 var blocked = []byte("[Bloqueada (inativa)]")
15
```

```
16 type coursesCache struct {
17         cache pool.StringCache
18 }
19
20 func prettify(data []byte) []byte {
21         return bytes.TrimSpace(data)
22 }
23
24 func stringify(data []byte) string {
25         words := bytes.Split(bytes.ToLower(prettify(data)), []byte(" "))
26         for i := range words {
27                 if len(words[i]) > 2 {
28                         words[i] = bytes.Title(words[i])
29                 }
30         }
31         return string(bytes.Join(words, []byte(" ")))
32 }
33
34 func cleanDiscipline(data []byte) string {
35         result := prettify(data)
36         return string(bytes.TrimSuffix(result, reserved))
37 }
38
39 func (cc *coursesCache) ParseName(data []byte) (newDiscipline, newCourse
   ↪  string, err error) {
40         lastIndex := bytes.LastIndex(data, []byte("*"))
41         err = errCourseNameNotFound
42         if lastIndex != -1 {
43                 newDiscipline = cc.cache.Transform(data[:lastIndex],
                  ↪  cleanDiscipline)
44                 newCourse = cc.cache.Transform(data[lastIndex+1:],
                  ↪  stringify)
45                 err = nil
46         }
47         return
48 }
49
50 func (cc *coursesCache) ParseID(data []byte) (result string, err error) {
51         startIndex := bytes.LastIndex(data, []byte("curso="))
```

```
52          err = errCourseIdNotFound
53          if startIndex != -1 {
54                  data = data[startIndex+6:]
55                  lastIndex := bytes.LastIndex(data, []byte("&"))
56                  if lastIndex != -1 {
57                          data = data[:lastIndex]
58                  }
59                  result = cc.cache.String(data)
60                  err = nil
61          }
62          return
63  }
64  func newCoursesCache(cac pool.StringCache) *coursesCache {
65          return &coursesCache{
66                  cache: cac,
67          }
68  }
```

Arquivo fetcher.go

```
1   package ufsc2offers
2
3   import (
4           "bytes"
5           "encoding/hex"
6           "fmt"
7           "hash"
8           "io"
9           "net/http"
10          "net/url"
11          "strconv"
12          "strings"
13          "time"
14
15          "github.com/davecgh/go-spew/spew"
16          "github.com/fjorgemota/gurudamatricula/robot/format"
17          "github.com/fjorgemota/gurudamatricula/robot/util/attrs"
18          "github.com/fjorgemota/gurudamatricula/util/clock"
19          "github.com/fjorgemota/gurudamatricula/util/kv"
20          "github.com/fjorgemota/gurudamatricula/util/pool"
```

```
21          "github.com/pkg/errors"
22
23          "golang.org/x/net/html"
24          "golang.org/x/net/html/atom"
25  )
26
27  // Stores the URL used to get the initial data
28  const initialURL =
    ↪  "https://cagr.sistemas.ufsc.br/modules/comunidade/cadastroTurmas/"
29
30  // Stores the URL used to get the data
31  const dataURL =
    ↪  "https://cagr.sistemas.ufsc.br/modules/comunidade/cadastroTurmas/index.xhtml"
32
33  // ErrAlreadyFetched is returned when the fetcher already catched that
    ↪  page
34  // but without other parameters, indicating a error in CAGR
35  var ErrAlreadyFetched = errors.New("ufsc: Fetcher already captured that
    ↪  page with other parameters")
36
37  // ErrInvalidStatus is returned when the HTTP Status returned by the
    ↪  server is
38  // not on the range between 200 and 299 (including these numbers)
39  var ErrInvalidStatus = errors.New("ufsc2: Invalid HTTP Status detected")
40
41  // ErrAlreadyDefined is returned when BootRequest is called but the
    ↪  session already
42  // has a view state defined
43  var ErrAlreadyDefined = errors.New("ufsc: viewState already defined")
44
45  // ErrViewStateNotFound is returned when the view state is not returned
    ↪  on a boot
46  // request
47  var ErrViewStateNotFound = errors.New("ufsc: ViewState not found")
48
49  // ErrNotBooted is returned when the user call Fetch without calling
    ↪  BootRequest first
50  var ErrNotBooted = errors.New("ufsc: Need to call BootRequest first")
51
```

```
52 type fetcherReader struct {
53         response io.ReadCloser
54         reader   io.Reader
55         hasher   hash.Hash
56         store    kv.Store
57         table    string
58         state    fetcherState
59 }
60
61 func (fr *fetcherReader) Read(b []byte) (int, error) {
62         n, err := fr.reader.Read(b)
63         if err == io.EOF {
64                 err = fr.response.Close()
65                 if err == nil && fr.store != nil && fr.table != "-" {
66                         hash := hex.EncodeToString(fr.hasher.Sum(nil))
67                         err = fr.store.Transaction([]kv.Reference{{Table:
                          ↪  fr.table, Key: hash}}, func(store
                          ↪  kv.LimitedStore) error {
68                                 var state fetcherState
69                                 errOp := store.Get(fr.table, hash,
                                  ↪  &state)
70                                 if kv.IsKeyNotFoundError(errOp) {
71                                         // If the key is not found it is
                                          ↪  a good sign, not a bad sign
72                                         errOp = nil
73                                 }
74                                 if errOp == nil && state.ID ==
                                  ↪  fr.state.ID &&
                                  ↪  !fr.state.equals(state) {
75                                         errOp =
                                          ↪  errors.Wrap(ErrAlreadyFetched,
                                          ↪  spew.Sprintf(" - Original:
                                          ↪  '%v'  - Actual: '%v' ",
                                          ↪  fr.state, state))
76                                 }
77                                 if errOp == nil {
78                                         errOp = store.Set(fr.table, hash,
                                          ↪  &fr.state)
79                                 }
```

```
80                                  return errOp
81                          })
82                  }
83                  pool.PutSha1(fr.hasher)
84                  fr.hasher = nil
85                  fr.reader = nil
86                  if err == nil {
87                          // If there is no errors, just return EOF
88                          err = io.EOF
89                  }
90          }
91          return n, err
92  }
93
94  type fetcherState struct {
95          ID         string
96          Method     string
97          Period     format.Period
98          Campus     format.Campus
99          PageNumber int
100         Date       time.Time
101 }
102
103 func (s1 fetcherState) equals(s2 fetcherState) bool {
104         return s1.Method == s2.Method &&
105                 s1.Period == s2.Period &&
106                 s1.Campus == s2.Campus &&
107                 s1.PageNumber == s2.PageNumber
108 }
109
110 type bootResponse struct {
111         semesters []format.Period
112         campi     []format.Campus
113         viewState string
114 }
115
116 func getBaseRequest(method string, body io.Reader, userAgent string)
    ↪  (*http.Request, error) {
117         var baseURL string
```

```
118         if method == "GET" {
119                 baseURL = initialURL
120         } else {
121                 baseURL = dataURL
122         }
123         req, err := http.NewRequest(method, baseURL, body)
124         if err == nil && method == "POST" {
125                 req.Header.Add("Referer",
                    ↪ "https://cagr.sistemas.ufsc.br/modules/cadastroTurmas/cadastroT
126                 req.Header.Add("Content-Type",
                    ↪ "application/x-www-form-urlencoded; charset=UTF-8")
127         }
128         if err == nil {
129                 req.Header.Add("User-Agent", userAgent)
130                 req.Header.Add("Pragma", "no-cache")
131                 req.Header.Add("Cache-Control", "no-cache")
132         }
133         return req, err
134 }
135
136 func getData(s format.Period, c format.Campus, p int, viewState string)
    ↪ url.Values {
137         data := url.Values{}
138         data.Set("AJAXREQUEST", "_viewRoot")
139         data.Set("AJAX:EVENTS_COUNT", "1")
140         data.Set("formBusca:selectSemestre", s.ID)
141         data.Set("formBusca:selectDepartamento", "")
142         data.Set("formBusca:selectCampus", c.ID)
143         data.Set("formBusca:selectCursosGraduacao", "0")
144         data.Set("formBusca:codigoDisciplina", "")
145         data.Set("formBusca:j_id100_selection", "")
146         data.Set("formBusca:filterDisciplina", "")
147         data.Set("formBusca:j_id104", "")
148         data.Set("formBusca:j_id119", "formBusca:j_id119")
149         data.Set("formBusca:j_id108_selection", "")
150         data.Set("formBusca:filterProfessor", "")
151         data.Set("formBusca:selectDiaSemana", "0")
152         data.Set("formBusca:selectHorarioSemana", "")
153         data.Set("formBusca", "formBusca")
```

```
154         data.Set("autoScroll", "")
155         data.Set("javax.faces.ViewState", viewState)
156         data.Set("isoladas", "")
157         data.Set("formBusca:dataScroller1", strconv.Itoa(p))
158         return data
159 }
160
161 func getDoc(state fetcherState, store kv.Store, checkTable string,
    ↪  response *http.Response) (*html.Tokenizer, error) {
162         var tok *html.Tokenizer
163         err := ErrInvalidStatus
164         if response.StatusCode >= 200 && response.StatusCode <= 299 {
165                 resp := response.Body
166                 hasher := pool.GetSha1()
167                 fr := &fetcherReader{
168                         hasher:   hasher,
169                         response: resp,
170                         reader:   io.TeeReader(resp, hasher),
171                         state:    state,
172                         store:    store,
173                         table:    checkTable,
174                 }
175                 tok = html.NewTokenizer(fr)
176                 err = nil
177         }
178         return tok, err
179 }
180
181 func bootRequest(client *http.Client, store kv.Store, table string, clk
    ↪  clock.Clock, lastNumberOfPeriods int, ID, userAgent string)
    ↪  (bootResponse, error) {
182         request, err := getBaseRequest("GET", nil, userAgent)
183         var response *http.Response
184         if err == nil {
185                 response, err = client.Do(request)
186         }
187         var doc *html.Tokenizer
188         if err == nil {
189                 doc, err = getDoc(fetcherState{
```

```
                        Method: "GET",
                        ID:     ID,
                        Date:   clk.Now(),
                }, store, table, response)
        }
        var exists bool
        var result bootResponse
        if err == nil {
                startCampi := false
                startCampus := false
                startPeriods := false
                startPeriod := false
                var campus format.Campus
                var semester format.Period
                attrTmp := pool.GetBytesBuffer()
                tmp := pool.GetBytesBuffer()
                values := make(map[string]string, 2)
                for err == nil {
                        tagType := doc.Next()
                        if tagType == html.ErrorToken {
                                err = doc.Err()
                        } else if tagType == html.SelfClosingTagToken &&
                        ↪  !exists {
                                tag, hasNextAttr := doc.TagName()
                                if atom.String(tag) == "input" {
                                        for key := range values {
                                                delete(values, key)
                                        }
                                        values, err = attrs.GetAttrs(doc,
                                        ↪  map[string]struct{}{"name":
                                        ↪  struct{}{}, "value":
                                        ↪  struct{}{}}, values,
                                        ↪  hasNextAttr)
                                        if err == nil && values["name"]
                                        ↪  == "javax.faces.ViewState" {
                                                result.viewState =
                                                ↪  values["value"]
                                                exists = true
                                        }
```

```
222                     }
223                 } else if tagType == html.StartTagToken {
224                     tag, hasNextAttr := doc.TagName()
225                     tagName := atom.String(tag)
226                     switch tagName {
227                     case "select":
228                         var value []byte
229                         value, err =
                            ↪ attrs.GetAttrBytes(doc, "id",
                            ↪ hasNextAttr, attrTmp)
230                         if bytes.Equal(value,
                            ↪ []byte("formBusca:selectSemestre"))
                            ↪ {
231                             startPeriods = true
232                             startPeriod = false
233                         } else if bytes.Equal(value,
                            ↪ []byte("formBusca:selectCampus"))
                            ↪ {
234                             startCampi = true
235                         }
236                     case "option":
237                         if startPeriods &&
                            ↪ len(result.semesters) <
                            ↪ lastNumberOfPeriods {
238                             if !startPeriod {
239                                 semester =
                                    ↪ format.Period{}
240                                 startPeriod =
                                    ↪ true
241                             }
242                             var value []byte
243                             value, err =
                                ↪ attrs.GetAttrBytes(doc,
                                ↪ "value", hasNextAttr,
                                ↪ attrTmp)
244                             if err == nil &&
                                ↪ len(value) > 0 {
245                                 semester.ID =
                                    ↪ string(value)
```

```
246                                     }
247                                 } else if startCampi {
248                                     if !startCampus {
249                                         campus =
                                            ↪ format.Campus{}
250                                         startCampus =
                                            ↪ true
251                                     }
252                                     var value []byte
253                                     value, err =
                                        ↪ attrs.GetAttrBytes(doc,
                                        ↪ "value", hasNextAttr,
                                        ↪ attrTmp)
254                                     if err == nil &&
                                        ↪ len(value) > 0 {
255                                         campus.ID =
                                            ↪ string(value)
256                                     }
257                                 }
258                             }
259                         } else if tagType == html.EndTagToken {
260                             tag, _ := doc.TagName()
261                             tagName := atom.String(tag)
262                             switch tagName {
263                             case "option":
264                                 if startPeriod {
265                                     text := tmp.Bytes()
266                                     ltext := len(text)
267                                     if bytes.Equal(text,
                                        ↪ []byte("2062")) ||
                                        ↪ (ltext == 5 &&
                                        ↪ semester.ID != "") {
268                                         semester.Name =
                                            ↪ fmt.Sprintf("%s-%c",
                                            ↪ text[:ltext-1],
                                            ↪ text[ltext-1])
```

```
269                                             result.semesters
                                                ↪  =
                                                ↪  append(result.semesters,
                                                ↪  semester)
270                                         } else {
271                                             err =
                                                ↪  fmt.Errorf("ufsc2:
                                                ↪  Unrecognized
                                                ↪  string
                                                ↪  detected on
                                                ↪  semester
                                                ↪  field '%s'",
                                                ↪  text)
272                                         }
273                                         startPeriod = false
274                                     } else if startCampus {
275                                         if campus.ID != "0" &&
                                            ↪  campus.ID != "" {
276                                             bs := tmp.Bytes()
277                                             if
                                                ↪  bytes.Equal(bs[:5],
                                                ↪  []byte("UFSC/"))
                                                ↪  {
278                                                 bs =
                                                    ↪  bs[5:]
279                                             }
280                                             campus.Name =
                                                ↪  string(bs)
281                                             result.campi =
                                                ↪  append(result.campi,
                                                ↪  campus)
282                                         }
283                                         startCampus = false
284                                     }
285                                     tmp.Reset()
286                                 case "select":
287                                     startPeriods = false
288                                     startCampi = false
289                                 }
```

```
290                         } else if tagType == html.TextToken &&
                            ↪  (startPeriod || startCampus) {
291                                 tmp.Write(doc.Text())
292                         }
293                 }
294                 pool.PutBytesBuffer(attrTmp)
295                 pool.PutBytesBuffer(tmp)
296         }
297         if err == io.EOF || err == ErrAlreadyFetched {
298                 // Can ignore EOF and ErrAlreadyFetched on the first page
299                 err = nil
300         }
301         if err == nil && !exists {
302                 return result, ErrViewStateNotFound
303         }
304         return result, err
305 }
306
307 func fetch(client *http.Client, store kv.Store, table string, clk
    ↪  clock.Clock, s format.Period, c format.Campus, pageNumber int,
    ↪  viewState string, ID, userAgent string) (*html.Tokenizer, error) {
308         err := ErrNotBooted
309         var doc *html.Tokenizer
310         if len(viewState) > 0 {
311                 data := getData(s, c, pageNumber, viewState)
312                 var request *http.Request
313                 request, err = getBaseRequest("POST",
                    ↪  strings.NewReader(data.Encode()), userAgent)
314                 var response *http.Response
315                 if err == nil {
316                         response, err = client.Do(request)
317                 }
318                 if err == nil {
319                         doc, err = getDoc(fetcherState{
320                                 Method:     "POST",
321                                 Period:     s,
322                                 Campus:     c,
323                                 PageNumber: pageNumber,
324                                 ID:         ID,
```

```
325                                 Date:       clk.Now(),
326                         }, store, table, response)
327                 }
328         }
329         return doc, err
330 }
```

Arquivo joiner.go

```
1  package ufsc2offers
2
3  import (
4          "io"
5
6          "github.com/fjorgemota/gurudamatricula/util/coder"
7  )
8
9  type handleJoiner struct {
10         final  bool
11         handle Handler
12 }
13
14 func (j handleJoiner) GetStreamingEncoder(writer io.WriteCloser)
   ↪  coder.StreamingEncoder {
15         if j.final {
16                 return j.handle.GetStreamingEncoder(writer)
17         }
18         return coder.NewGOBStreamingEncoder(writer)
19 }
20
21 func (j handleJoiner) GetStreamingDecoder(reader io.Reader)
   ↪  coder.StreamingDecoder {
22         return coder.NewGOBStreamingDecoder(reader)
23 }
```

Arquivo options.go

```
1  package ufsc2offers
2
```

```
3  import "time"
4
5  type Tasks struct {
6          FetchCampus string
7          DeleteFiles string
8          JoinFiles   string
9          Start       string
10 }
11
12 // Options define some Options that should be defined in compile-time
13 type Options struct {
14         Tasks                Tasks
15         BatchSize            int
16         MaxRetries           int
17         NumberOfPeriods      int
18         DelayBetweenRequests time.Duration
19         CheckTable           string
20         UserAgent            string
21 }
22
23 func getDefaults(opt Options) Options {
24         if opt.Tasks.FetchCampus == "" {
25                 opt.Tasks.FetchCampus = "ufsc.fetch"
26         }
27         if opt.Tasks.JoinFiles == "" {
28                 opt.Tasks.JoinFiles = "ufsc.join"
29         }
30         if opt.Tasks.DeleteFiles == "" {
31                 opt.Tasks.DeleteFiles = "ufsc.delete"
32         }
33         if opt.Tasks.Start == "" {
34                 opt.Tasks.Start = "ufsc.start"
35         }
36         if opt.CheckTable == "" {
37                 opt.CheckTable = "ufsc_check"
38         }
39         if opt.BatchSize <= 0 {
40                 opt.BatchSize = 10
41         }
```

```
42          if opt.MaxRetries == 0 {
43                  opt.MaxRetries = 3
44          }
45          if opt.UserAgent == "" {
46                  opt.UserAgent = "Mozilla/5.0 (X11; Linux x86_64)
                    ↪  AppleWebKit/537.36 (KHTML, like Gecko)
                    ↪  Chrome/60.0.3112.101 Safari/537.36"
47          }
48          if opt.DelayBetweenRequests <= 0 {
49                  opt.DelayBetweenRequests = 2 * time.Second
50          }
51          if opt.MaxRetries < 0 {
52                  opt.MaxRetries = 0
53          }
54          if opt.NumberOfPeriods < 1 {
55                  opt.NumberOfPeriods = 3
56          }
57          return opt
58  }
```

Arquivo parser.go

```
1   package ufsc2offers
2
3   import (
4           "bytes"
5           "crypto/sha1"
6           "encoding/hex"
7           "io"
8
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/robot/util/attrs"
11          robotpool
            ↪  "github.com/fjorgemota/gurudamatricula/robot/format/pool"
12          "github.com/fjorgemota/gurudamatricula/util/pool"
13          "github.com/pkg/errors"
14          "github.com/sirupsen/logrus"
15          "golang.org/x/net/html"
16          "golang.org/x/net/html/atom"
17  )
```

```
18
19 func hashSha1(str string) string {
20         bs := sha1.Sum([]byte(str))
21         return hex.EncodeToString(bs[:])
22 }
23
24 func getVacancyTable() format.Table {
25         var table format.Table
26         table.ID = "vacancies"
27         table.Columns = []format.Column{{ID: "type", Name: "Tipo"}, {ID:
           ↪  "quantity", Name: "Quantidade"}}
28         return table
29 }
30
31 func parse(teams []*format.Team, doc *html.Tokenizer, logger
   ↪  logrus.FieldLogger, actualPeriod format.Period, actualCampus
   ↪  format.Campus, pageNumber int, stringCache pool.StringCache, intCache
   ↪  pool.AtoiCache, courseCache *coursesCache) ([]*format.Team, bool,
   ↪  error) {
32         startTeams := false
33         startPager := false
34         teamState := -1
35         hasNext := false
36         actualTeam := robotpool.GetTeam()
37         foundTeams := false
38         foundPager := false
39         tmp := pool.GetBytesBuffer()
40         disciplineName := pool.GetBytesBuffer()
41         vacancyTable := getVacancyTable()
42         var err error
43         var teacher format.Teacher
44         for err == nil {
45                 tagType := doc.Next()
46                 if tagType == html.ErrorToken {
47                         err = doc.Err()
48                 } else if tagType == html.StartTagToken {
49                         tag, hasNextAttr := doc.TagName()
50                         tagName := atom.String(tag)
51                         switch tagName {
```

```
52                      case "table":
53                              var value []byte
54                              value, err = attrs.GetAttrBytes(doc, "id",
                                ↪  hasNextAttr, tmp)
55                              logger.WithField("id",
                                ↪  string(value)).Debug("Found Table")
56                              startTeams = bytes.Equal(value,
                                ↪  []byte("formBusca:dataTable"))
57                              startPager = bytes.Equal(value,
                                ↪  []byte("formBusca:dataScroller1_table"))
58                              if startTeams && !foundTeams {
59                                      logger.Debug("Found data table!")
60                                      foundTeams = true
61                              } else if startTeams && foundTeams {
62                                      err = errors.New("ufsc2:
                                        ↪  dataTable found twice in the
                                        ↪  page")
63                              }
64                              if startPager && !foundPager {
65                                      logger.Debug("Found pager!")
66                                      foundPager = true
67                              } else if startPager && foundPager {
68                                      err = errors.New("ufsc2:
                                        ↪  dataScroller1_table found
                                        ↪  twice in the page")
69                              }
70                      case "tr":
71                              if startTeams {
72                                      var value []byte
73                                      value, err =
                                        ↪  attrs.GetAttrBytes(doc,
                                        ↪  "class", hasNextAttr, tmp)
74                                      if bytes.Contains(value,
                                        ↪  []byte("rich-table-row")) {
75                                              teamState = 0
76                                      }
77                              }
78                      case "td":
79                              if startPager == false {
```

```
80                                      continue
81                              }
82                              var value []byte
83                              value, err = attrs.GetAttrBytes(doc,
                                ↪  "onclick", hasNextAttr, tmp)
84                              if bytes.Contains(value, []byte("next"))
                                ↪  {
85                                      hasNext = true
86                              }
87                      case "a":
88                              switch teamState {
89                              case 2:
90                                      var value []byte
91                                      value, err =
                                        ↪  attrs.GetAttrBytes(doc,
                                        ↪  "href", hasNextAttr, tmp)
92                                      if err == nil {
93                                              actualTeam.Course.ID, err
                                                ↪  =
                                                ↪  courseCache.ParseID(value)
94                                      }
95                              case 13:
96                                      var value []byte
97                                      value, err =
                                        ↪  attrs.GetAttrBytes(doc,
                                        ↪  "href", hasNextAttr, tmp)
98                                      if err == nil {
99                                              teacher.URL =
                                                ↪  stringCache.String(value)
100                                     }
101                             }
102                     }
103             } else if tagType == html.EndTagToken {
104                     tag, _ := doc.TagName()
105                     tagName := atom.String(tag)
106                     switch tagName {
107                     case "table":
108                             startPager = false
109                             startTeams = false
```

```
                        case "tr":
                                if teamState != -1 {
                                        disciplineNameBytes :=
                                        ↪  disciplineName.Bytes()
                                        if
                                        ↪  !bytes.Contains(disciplineNameBytes,
                                        ↪  cancelled) &&
                                        ↪  !bytes.Contains(disciplineNameBytes,
                                        ↪  blocked) {
                                                if
                                                ↪  bytes.Contains(disciplineNameBytes,
                                                ↪  reserved) {
                                                        actualTeam.Course.Exclusive
                                                        ↪  =
                                                        ↪  format.Exclusive
                                                        disciplineNameBytes
                                                        ↪  =
                                                        ↪  bytes.Replace(disciplineName
                                                        ↪  reserved,
                                                        ↪  []byte(""),
                                                        ↪  -1)
                                                } else {
                                                        actualTeam.Course.Exclusive
                                                        ↪  =
                                                        ↪  format.Unknown
                                                }
                                                actualTeam.Discipline.Name,
                                                ↪  actualTeam.Course.Name,
                                                ↪  err =
                                                ↪  courseCache.ParseName(disciplineName
                                                actualTeam.Campus =
                                                ↪  actualCampus
                                                actualTeam.Period =
                                                ↪  actualPeriod
                                                actualTeam.Tables =
                                                ↪  append(actualTeam.Tables,
                                                ↪  vacancyTable)
                                                teams = append(teams,
                                                ↪  actualTeam)
```

```
125                                     }
126                                     disciplineName.Reset()
127                                     teamState = -1
128                                     actualTeam = robotpool.GetTeam()
129                                     vacancyTable = getVacancyTable()
130                             }
131                     case "td":
132                             if teamState > -1 {
133                                     teamState++
134                             }
135                             if teamState == 13 {
136                                     teacher = format.Teacher{}
137                             }
138                     }
139             } else if tagType == html.TextToken && teamState != -1 {
140                     switch teamState {
141                     case 3:
142                             actualTeam.Discipline.Code =
                                ↪ stringCache.String(doc.Text())
143                             actualTeam.Discipline.ID =
                                ↪ hashSha1(actualTeam.Discipline.Code)
144                             actualTeam.Discipline.GenericID =
                                ↪ actualTeam.Discipline.ID
145                     case 4:
146                             actualTeam.Code =
                                ↪ stringCache.String(doc.Text())
147                             actualTeam.ID = hashSha1(actualTeam.Code)
148                     case 5:
149                             if disciplineName.Len() > 0 {
150                                     disciplineName.WriteRune(' ')
151                             }
152                             disciplineName.Write(doc.Text())
153                     case 7:
154                             actualTeam.Vacancies.Offered, err =
                                ↪ intCache.Atoi(doc.Text())
155                     case 8:
156                             actualTeam.Vacancies.Filled, err =
                                ↪ intCache.Atoi(doc.Text())
157                             if actualTeam.Vacancies.Filled <= 0 {
```

```
158                                     actualTeam.Vacancies.Filled = 0
159                             }
160                     case 9:
161                             vacancyTable.Rows =
                                ↪  append(vacancyTable.Rows,
                                ↪  map[string]string{"type": "Estudantes
                                ↪  Especiais", "quantity":
                                ↪  stringCache.String(doc.Text())})
162                     case 10:
163                             vacancyTable.Rows =
                                ↪  append(vacancyTable.Rows,
                                ↪  map[string]string{"type": "Balanço de
                                ↪  Vagas", "quantity":
                                ↪  stringCache.String(doc.Text())})
164                     case 11:
165                             vacancyTable.Rows =
                                ↪  append(vacancyTable.Rows,
                                ↪  map[string]string{"type": "Sem Vagas",
                                ↪  "quantity":
                                ↪  stringCache.String(doc.Text())})
166                     case 12:
167                             var schedule format.Schedule
168                             schedule, err =
                                ↪  parseScheduleText(doc.Text(),
                                ↪  intCache.Atoi)
169                             if err == nil {
170                                     actualTeam.Schedules =
                                        ↪  append(actualTeam.Schedules,
                                        ↪  schedule)
171                             }
172                     case 13:
173                             teacherName := doc.Text()
174                             teacher.Name =
                                ↪  stringCache.String(teacherName)
175                             teacher.ID = hashSha1(teacher.Name)
176                             if err == nil {
177                                     actualTeam.Teachers =
                                        ↪  append(actualTeam.Teachers,
                                        ↪  teacher)
```

```
178                                     }
179                             }
180                     }
181             }
182             pool.PutBytesBuffer(disciplineName)
183             pool.PutBytesBuffer(tmp)
184             if err == io.EOF {
185                     err = nil
186             }
187             logger.WithField("found_teams",
        ↪  foundTeams).WithField("found_pager",
        ↪  foundPager).Debug("Here's what I found")
188             if pageNumber > 1 {
189                     // These errors just do not make sense in the first page
190                     if !foundTeams && err == nil {
191                             logger.Error("Data table not found!")
192                             err = errors.New("ufsc2: dataTable not found in
                            ↪  the page")
193                     }
194                     if !foundPager && err == nil {
195                             logger.Error("Pager not found!")
196                             err = errors.New("ufsc2: dataScroller1_table not
                            ↪  found in the page")
197                     }
198             } else if errors.Cause(err) == ErrAlreadyFetched && len(teams) ==
        ↪  0 {
199                     // Ignore that error on the first page
200                     err = nil
201             }
202             if err != nil {
203                     logger.WithError(err).Error("error found while parsing
                    ↪  page")
204             }
205             return teams, hasNext, err
206     }
```

Arquivo schedule.go

```
1   package ufsc2offers
2
```

```
3  import (
4          "errors"
5          "fmt"
6          "sort"
7
8          "github.com/fjorgemota/gurudamatricula/robot/format"
9  )
10
11 var errScheduleEmpty = errors.New("schedule string cannot be empty")
12
13 type ufscHour struct {
14         Hour   int8
15         Minute int8
16 }
17
18 type ufscClass struct {
19         Start ufscHour
20         End   ufscHour
21 }
22
23 var ufscHours = []ufscClass{
24         {Start: ufscHour{Hour: 7, Minute: 30}, End: ufscHour{Hour: 8,
           ↪  Minute: 20}},
25         {Start: ufscHour{Hour: 8, Minute: 20}, End: ufscHour{Hour: 9,
           ↪  Minute: 10}},
26         {Start: ufscHour{Hour: 9, Minute: 10}, End: ufscHour{Hour: 10,
           ↪  Minute: 0}},
27         {Start: ufscHour{Hour: 10, Minute: 10}, End: ufscHour{Hour: 11,
           ↪  Minute: 0}},
28         {Start: ufscHour{Hour: 11, Minute: 0}, End: ufscHour{Hour: 11,
           ↪  Minute: 50}},
29         {Start: ufscHour{Hour: 13, Minute: 30}, End: ufscHour{Hour: 14,
           ↪  Minute: 20}},
30         {Start: ufscHour{Hour: 14, Minute: 20}, End: ufscHour{Hour: 15,
           ↪  Minute: 10}},
31         {Start: ufscHour{Hour: 15, Minute: 10}, End: ufscHour{Hour: 16,
           ↪  Minute: 0}},
32         {Start: ufscHour{Hour: 16, Minute: 20}, End: ufscHour{Hour: 17,
           ↪  Minute: 10}},
```

```
33          {Start: ufscHour{Hour: 17, Minute: 10}, End: ufscHour{Hour: 18,
            ↪  Minute: 0}},
34          {Start: ufscHour{Hour: 18, Minute: 30}, End: ufscHour{Hour: 19,
            ↪  Minute: 20}},
35          {Start: ufscHour{Hour: 19, Minute: 20}, End: ufscHour{Hour: 20,
            ↪  Minute: 10}},
36          {Start: ufscHour{Hour: 20, Minute: 20}, End: ufscHour{Hour: 21,
            ↪  Minute: 10}},
37          {Start: ufscHour{Hour: 21, Minute: 10}, End: ufscHour{Hour: 22,
            ↪  Minute: 0}},
38  }
39
40  func detectEnd(hourStart, minuteStart, numberOfLessons int8) (ufscHour,
    ↪  error) {
41          ind := sort.Search(len(ufscHours), func(n int) bool {
42                  return ufscHours[n].Start.Hour >= hourStart
43          })
44          var end ufscHour
45          var err error
46          err = fmt.Errorf("Invalid hour for a class with hourStart=%d,
            ↪  minuteStart=%d and numberOfLessons=%d", hourStart,
            ↪  minuteStart, numberOfLessons)
47          if ind < len(ufscHours) && ufscHours[ind].Start.Hour == hourStart
            ↪  && ufscHours[ind].Start.Minute == minuteStart {
48                  endInd := ind + int(numberOfLessons) - 1
49                  err = fmt.Errorf("Invalid end for a class with
                    ↪  hourStart=%d, minuteStart=%d and numberOfLessons=%d",
                    ↪  hourStart, minuteStart, numberOfLessons)
50                  if endInd < len(ufscHours) {
51                          end = ufscHours[endInd].End
52                          err = nil
53                  }
54          }
55          return end, err
56
57  }
58  func parseScheduleText(schedText []byte, Atoi func([]byte) (int, error))
    ↪  (format.Schedule, error) {
59          var sched format.Schedule
```

```
60          err := errScheduleEmpty
61          if len(schedText) > 0 {
62                  var dayOfWeek, hourStart, minuteStart, numberOfLessons
                    ↪  int
63                  dayOfWeek, err = Atoi(schedText[:1])
64                  if err == nil {
65                          hourStart, err = Atoi(schedText[2:4])
66                  }
67                  if err == nil {
68                          minuteStart, err = Atoi(schedText[4:6])
69                  }
70                  if err == nil {
71                          numberOfLessons, err = Atoi(schedText[7:8])
72                  }
73                  if err == nil {
74                          sched.DayOfWeek = int8(dayOfWeek)
75                          sched.Start.Hour = int8(hourStart)
76                          sched.Start.Minute = int8(minuteStart)
77                          room := string(schedText[11:])
78                          sched.Room.ID = room
79                          sched.Room.Name = room
80                  }
81                  var end ufscHour
82                  if err == nil {
83                          end, err = detectEnd(sched.Start.Hour,
                            ↪  sched.Start.Minute, int8(numberOfLessons))
84                  }
85                  if err == nil {
86                          sched.End.Hour = end.Hour
87                          sched.End.Minute = end.Minute
88                  }
89          }
90          return sched, err
91  }
```

Arquivo state.go

```
1  package ufsc2offers
2
3  import (
```

```
4           "bytes"
5           "io"
6
7           "github.com/fjorgemota/gurudamatricula/robot/format"
8           "github.com/fjorgemota/gurudamatricula/robot/util/ccookiejar"
9           "github.com/fjorgemota/gurudamatricula/util/coder"
10          "github.com/fjorgemota/gurudamatricula/util/pool"
11  )
12
13  type State struct {
14          ID          string
15          Period      format.Period
16          Campus      format.Campus
17          PageNumber  int
18          ViewState   string
19          Retries     int
20          Cookies     ccookiejar.CodedCookieJar
21          Prefix      string
22          Names       []string
23  }
24
25  type encodedState struct {
26          ID          string
27          Period      format.Period
28          Campus      format.Campus
29          PageNumber  int
30          Retries     int
31          ViewState   string
32          Prefix      string
33          Names       []string
34          Cookies     []byte
35  }
36
37  func (s *State) EncodeTo(cod coder.Coder, writer io.Writer) error {
38          var err error
39          encoded := encodedState{
40                  ID:          s.ID,
41                  Period:      s.Period,
42                  Campus:      s.Campus,
```

```
43                  PageNumber: s.PageNumber,
44                  ViewState:  s.ViewState,
45                  Retries:    s.Retries,
46                  Prefix:     s.Prefix,
47                  Names:      s.Names,
48          }
49          if s.Cookies != nil {
50                  buf := pool.GetBytesBuffer()
51                  defer pool.PutBytesBuffer(buf)
52                  err = cod.EncodeTo(s.Cookies, buf)
53                  if err == nil {
54                          encoded.Cookies = buf.Bytes()
55                  }
56          }
57          if err == nil {
58                  err = cod.EncodeTo(encoded, writer)
59          }
60          return err
61  }
62
63  func (s *State) DecodeFrom(cod coder.Coder, source io.Reader) error {
64          var encoded encodedState
65          err := cod.DecodeFrom(source, &encoded)
66          if err == nil {
67                  s.ID = encoded.ID
68                  s.Period = encoded.Period
69                  s.Campus = encoded.Campus
70                  s.PageNumber = encoded.PageNumber
71                  s.ViewState = encoded.ViewState
72                  s.Retries = encoded.Retries
73                  s.Prefix = encoded.Prefix
74                  s.Names = encoded.Names
75                  if encoded.Cookies != nil {
76                          err =
                            ↪  cod.DecodeFrom(bytes.NewReader(encoded.Cookies),
                            ↪  s.Cookies)
77                  }
78          }
79          return err
```

```
}

var _ coder.Encoder = &State{}
var _ coder.Decoder = &State{}
```

B.1.17 Pasta robot/usp2habilitations

Arquivo api.go

```
package usp2habilitations

import (
        "fmt"
        "math/rand"
        "net/http"
        "time"

        "github.com/fjorgemota/gurudamatricula/robot/format"
        "github.com/fjorgemota/gurudamatricula/util/clock"
        "github.com/fjorgemota/gurudamatricula/util/coder"
        "github.com/fjorgemota/gurudamatricula/util/pool"
        "github.com/sirupsen/logrus"
)

func randomSleep(clk clock.Clock, delay time.Duration) {
        clk.Sleep(time.Duration((rand.Float64() + 0.5) * float64(delay)))
}

func FetchCampi(logger logrus.FieldLogger, client *http.Client, clk
↪  clock.Clock, options BaseOptions, campi []format.Campus, ID,
↪  userAgent string) ([]State, error) {
        var result []State
        var err error
        stringCache := pool.GetStringCache()
        defer pool.PutStringCache(stringCache)
        for _, campus := range campi {
                var oldHabilitations, habilitations []format.Habilitation
                if err == nil {
                        randomSleep(clk, options.DelayBetweenRequests)
```

```
29                      habilitations, err = GetHabililitations(client,
                        ↪  stringCache, campus, ID, userAgent)
30              }
31              filteredHabilitations := habilitations[:0]
32              habIDs := make(map[habilitationId]struct{},
                ↪  len(habilitations))
33              for _, hab := range habilitations {
34                      habID := toHabilitationID(hab)
35                      if _, ok := habIDs[habID]; !ok {
36                              filteredHabilitations =
                                ↪  append(filteredHabilitations, hab)
37                      }
38                      habIDs[habID] = struct{}{}
39              }
40              logger.WithField("num_habilitations",
                ↪  len(filteredHabilitations)).WithField("campus",
                ↪  campus.Name).Debug("Recognizing habilitations")
41              if err == nil {
42                      result = append(result, State{
43                              Habilitations: filteredHabilitations,
44                              Campus:        campus,
45                              ID:            ID,
46                              Prefix:        fmt.Sprintf("%s-%d",
                                ↪  campus.ID, clk.Now().Unix()),
47                      })
48              }
49              if err == nil {
50                      oldHabilitations, err =
                        ↪  GetOldHabililitations(client, stringCache,
                        ↪  campus, ID, userAgent)
51              }
52              filteredOldHabilitations := oldHabilitations[:0]
53              for _, hab := range oldHabilitations {
54                      habID := toHabilitationID(hab)
55                      if _, ok := habIDs[habID]; !ok {
56                              filteredOldHabilitations =
                                ↪  append(filteredOldHabilitations, hab)
57                      }
58                      habIDs[habID] = struct{}{}
```

```
59                  }
60                  logger.WithField("num_old_habilitations",
                    ↪  len(filteredOldHabilitations)).WithField("campus",
                    ↪  campus.Name).Debug("Recognizing old habilitations")
61                  if err == nil {
62                          result = append(result, State{
63                                  Habilitations: filteredOldHabilitations,
64                                  Campus:        campus,
65                                  ID:            ID,
66                                  Prefix:        fmt.Sprintf("%s-%d",
                                    ↪  campus.ID, clk.Now().Unix()),
67                          })
68                  }
69          }
70          return result, err
71  }
72
73  func BalanceHabilitations(logger logrus.FieldLogger, clk clock.Clock,
    ↪  parallelRequests int, dispatch func([]State) error, states
    ↪  ...[]State) error {
74          var allStates []State
75          for _, state := range states {
76                  allStates = append(allStates, state...)
77          }
78          habilitationsCount := 0
79          for _, state := range allStates {
80                  habilitationsCount += len(state.Habilitations)
81          }
82          var batch []State
83          var actualState State
84          splitPoint := habilitationsCount / parallelRequests
85          logger.WithField("habilitations_count",
            ↪  habilitationsCount).WithField("splitPoint",
            ↪  splitPoint).Debug("Detecting habilitations to balance")
86          count := 0
87          var err error
88          for _, selectedState := range allStates {
89                  if selectedState.Campus != actualState.Campus {
90                          if len(actualState.Habilitations) > 0 {
```

```
91                                  batch = append(batch, actualState)
92                          }
93                          actualState = State{}
94                          actualState.Campus = selectedState.Campus
95                          actualState.ID = selectedState.ID
96                          actualState.Prefix = selectedState.Prefix
97                  }
98                  for _, habilitation := range selectedState.Habilitations
                    ↪ {
99                          if count > splitPoint && err == nil {
100                                 batch = append(batch, actualState)
101                                 err = dispatch(batch)
102                                 batch = batch[:0]
103                                 actualState = State{}
104                                 actualState.Campus = selectedState.Campus
105                                 actualState.ID = selectedState.ID
106                                 actualState.Prefix = selectedState.Prefix
107                                 count = 0
108                         }
109                         actualState.Habilitations =
                            ↪  append(actualState.Habilitations,
                            ↪  habilitation)
110                         count++
111                 }
112         }
113         if count > 0 && err == nil {
114                 batch = append(batch, actualState)
115                 err = dispatch(batch)
116         }
117         return err
118 }
119
120 func FetchDiscipline(client *http.Client, clk clock.Clock, options
    ↪  BaseOptions, actualState DisciplineState) error {
121         randomSleep(clk, options.DelayBetweenRequests)
122         options = getBaseDefaults(options)
123         disciplineReader, err := fetchDiscipline(client,
            ↪  actualState.Discipline, actualState.Habilitation,
            ↪  actualState.ID, options.UserAgent)
```

```
124         if err == nil {
125                 stringCache := actualState.StringCache
126                 if stringCache == nil {
127                         stringCache = pool.GetStringCache()
128                         defer pool.PutStringCache(stringCache)
129                 }
130                 intCache := actualState.IntCache
131                 if intCache == nil {
132                         intCache = pool.GetAtoiCache()
133                         defer pool.PutAtoiCache(intCache)
134                 }
135                 err = parseDisciplinePage(actualState.Discipline,
                    ↪  actualState.Habilitation, disciplineReader, intCache,
                    ↪  stringCache)
136         }
137         return err
138 }
139
140 func FetchHabilitation(client *http.Client, encoder
    ↪  coder.StreamingEncoder, clk clock.Clock, options HabilitationOptions,
    ↪  actualState HabilitationState) error {
141         randomSleep(clk, options.Base.DelayBetweenRequests)
142         reader, err := fetchHabilitation(client, actualState.Campus,
            ↪  actualState.Habilitation, actualState.ID,
            ↪  options.Base.UserAgent)
143         stringCache := actualState.StringCache
144         intCache := actualState.IntCache
145         if err == nil && stringCache == nil {
146                 stringCache = pool.GetStringCache()
147                 defer pool.PutStringCache(stringCache)
148         }
149         if err == nil && intCache == nil {
150                 intCache = pool.GetAtoiCache()
151                 defer pool.PutAtoiCache(intCache)
152         }
153         hab := actualState.Habilitation
154         if err == nil {
155                 hab, err = parseHabilitationPage(hab, reader, intCache,
                    ↪  stringCache)
```

```
156         }
157         if err == nil && hab.Name != "" && hab.Course.Name != "" &&
            ↪  len(hab.Steps) > 0 {
158                 for stepIndex := range hab.Steps {
159                         if !options.GetDisciplines {
160                                 continue
161                         }
162                         disciplines := hab.Steps[stepIndex].Disciplines
163                         for disciplineIndex := range
                            ↪  hab.Steps[stepIndex].Disciplines {
164                                 if err == nil {
165                                         err = FetchDiscipline(client, clk,
                                            ↪  options.Base,
                                            ↪  DisciplineState{
166                                                 Habilitation: hab,
167                                                 Discipline:
                                                    ↪  &disciplines[disciplineIndex],
168                                                 ID:
                                                    ↪  actualState.ID,
169                                                 IntCache:     intCache,
170                                                 StringCache:
                                                    ↪  stringCache,
171                                         })
172                                 }
173                         }
174                 }
175                 if err == nil {
176                         err = encoder.Encode(&hab)
177                 }
178         }
179         return err
180 }
```

Arquivo bot.go

```
1  package usp2habilitations
2
3  import (
4          "crypto/sha1"
5          "fmt"
```

```
6           "io"
7           "net/http"
8
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/robot/joiner"
11          "github.com/fjorgemota/gurudamatricula/robot/util/uspreq"
12          "github.com/fjorgemota/gurudamatricula/util/clock"
13          "github.com/fjorgemota/gurudamatricula/util/file"
14          "github.com/fjorgemota/gurudamatricula/util/pipeline"
15          "github.com/fjorgemota/gurudamatricula/util/pool"
16          "github.com/sirupsen/logrus"
17          "golang.org/x/net/context"
18  )
19
20  // Handler defines a few constructors that Bot needs to work
21  type Handler interface {
22          joiner.Handler
23          GetHTTPClient(ctx context.Context, jar http.CookieJar)
            ↪  *http.Client
24          GetManager(context.Context) (file.Manager, error)
25          GetLogger(context.Context) logrus.FieldLogger
26  }
27
28  type bot struct {
29          options PipelineOptions
30          handler Handler
31          clk     clock.Clock
32  }
33
34  func (b bot) start(ctx context.Context, pipe pipeline.Pipeline, target
    ↪  string) (pipeline.FutureID, error) {
35          client := b.handler.GetHTTPClient(ctx, nil)
36          logger := b.handler.GetLogger(ctx)
37          var results []pipeline.FutureID
38          err := uspreq.Start(logger, client, func(campi []format.Campus,
            ↪  ID string) error {
39                  futIDs, errDispatch :=
                    ↪  pipe.DispatchWithOptions(b.options.Tasks.FetchCampi,
                    ↪  pipeline.TaskOptions{
```

```
40                      MaxAttempts: 1,
41              }, campi, ID)
42              if errDispatch == nil {
43                      results = append(results, futIDs[0])
44              }
45              return errDispatch
46      }, b.options.ParallelRequests, b.options.Crawler.Base.UserAgent)
47      if err == nil {
48              args := make([]interface{}, 0, len(results)+1)
49              args = append(args, target)
50              for _, result := range results {
51                      args = append(args, result)
52              }
53              results, err =
                ↪  pipe.Dispatch(b.options.Tasks.BalanceHabilitations,
                ↪  args...)
54      }
55      var result pipeline.FutureID
56      if err == nil {
57              result = results[0]
58      }
59      return result, err
60  }
61
62  func (b bot) Start(pipe pipeline.Pipeline, target string)
    ↪  (pipeline.FutureID, error) {
63      futIDs, err := pipe.DispatchWithOptions(b.options.Tasks.Start,
        ↪  pipeline.TaskOptions{
64              MaxAttempts: 1,
65      }, target)
66      var futID pipeline.FutureID
67      if len(futIDs) > 0 {
68              futID = futIDs[0]
69      }
70      return futID, err
71  }
72
```

```
73  func (b bot) balanceHabilitations(ctx context.Context, pipe
    ↪  pipeline.Pipeline, target string, states ...[]State)
    ↪  (pipeline.FutureID, error) {
74          logger := b.handler.GetLogger(ctx)
75          var results []pipeline.FutureID
76          err := BalanceHabilitations(logger, b.clk,
            ↪  b.options.ParallelRequests, func(batch []State) error {
77                  result, err :=
                    ↪  pipe.Dispatch(b.options.Tasks.FetchHabilitations,
                    ↪  batch)
78                  if err == nil {
79                          results = append(results, result[0])
80                  }
81                  return err
82          }, states...)
83          var result pipeline.FutureID
84          if err == nil {
85                  files := make([]interface{}, 0, len(results)+1)
86                  files = append(files, target)
87                  for _, result := range results {
88                          files = append(files, result)
89                  }
90                  results, err = pipe.Dispatch(b.options.Tasks.JoinFiles,
                    ↪  files...)
91          }
92          if err == nil {
93                  result = results[0]
94          }
95          return result, err
96  }
97
98  func (b bot) generateName(actualState State) string {
99          var target []byte
100         for index := 0; index < b.options.BatchSize && index <
            ↪  len(actualState.Habilitations); index++ {
101                 target = append(target,
                    ↪  actualState.Habilitations[index].ID...)
102         }
103         now := b.clk.Now()
```

```
104         return fmt.Sprintf("%s-%x-%d-%d.partial_1", actualState.Prefix,
            ↪  sha1.Sum(target), now.Unix(), now.Nanosecond())
105 }
106
107 func (b bot) joinFiles(ctx context.Context, pipe pipeline.Pipeline,
    ↪  target string, filenames ...string) (string, error) {
108         var err error
109         result := filenames[0]
110         if len(filenames) > 1 {
111                 var manager file.Manager
112                 manager, err = b.handler.GetManager(ctx)
113                 if err == nil {
114                         err = joiner.JoinHabilitations(manager, b.handler,
                            ↪  filenames, target)
115                 }
116                 if err == nil {
117                         result = target
118                 }
119         }
120         return result, err
121 }
122 func (b bot) fetchCampi(ctx context.Context, pipe pipeline.Pipeline,
    ↪  campi []format.Campus, ID string) ([]State, error) {
123         logger := b.handler.GetLogger(ctx)
124         client := b.handler.GetHTTPClient(ctx, nil)
125         return FetchCampi(logger, client, b.clk, b.options.Crawler.Base,
            ↪  campi, ID, b.options.Crawler.Base.UserAgent)
126 }
127
128 func (b bot) fetchHabilitations(ctx context.Context, pipe
    ↪  pipeline.Pipeline, states []State) (pipeline.FutureID, error) {
129         logger := b.handler.GetLogger(ctx)
130         client := b.handler.GetHTTPClient(ctx, nil)
131         err := ErrInvalidStates
132         if len(states) > 0 {
133                 err = nil
134         }
135         habilitationCount := 0
136         for _, state := range states {
```

```
137                 habilitationCount += len(state.Habilitations)
138         }
139         stateIndex := 0
140         for ; stateIndex < len(states); stateIndex++ {
141                 if len(states[stateIndex].Habilitations) > 0 {
142                         break
143                 }
144         }
145         logger.WithField("habilitation_count",
            ↪  habilitationCount).WithField("campus",
            ↪  states[stateIndex].Campus.Name).Debug("Processing
            ↪  habilitations")
146         name := b.generateName(states[stateIndex])
147         var manager file.Manager
148         if err == nil {
149                 manager, err = b.handler.GetManager(ctx)
150         }
151         var writer io.WriteCloser
152         if err == nil {
153                 writer, err = manager.Writer(name)
154         }
155         stringCache := pool.GetStringCache()
156         intCache := pool.GetAtoiCache()
157         defer pool.PutStringCache(stringCache)
158         defer pool.PutAtoiCache(intCache)
159         var index int
160         if err == nil {
161                 encoder := b.handler.GetStreamingEncoder(writer)
162                 actualState := states[stateIndex]
163                 for c := 0; err == nil && c < b.options.BatchSize &&
                    ↪  index < len(actualState.Habilitations); c++ {
164                         err = FetchHabilitation(client, encoder, b.clk,
                            ↪  b.options.Crawler, HabilitationState{
165                                 Campus:       actualState.Campus,
166                                 Habilitation:
                                    ↪  actualState.Habilitations[index],
167                                 ID:           actualState.ID,
168                                 IntCache:     intCache,
169                                 StringCache:  stringCache,
```

```
170                     })
171                     if err == nil {
172                             index++
173                     }
174             }
175             if err == nil {
176                     err = encoder.Flush()
177             }
178             if err == nil {
179                     err = encoder.Close()
180             }
181     }
182     shouldDispatch := false
183     if err == nil {
184             // Only add file if we're not able to complete this file
185             states[stateIndex].Names =
                ↪  append(states[stateIndex].Names, name)
186     } else {
187             states[stateIndex].Retries++
188             shouldDispatch = states[stateIndex].Retries <
                ↪  b.options.MaxRetries
189             if shouldDispatch {
190                     logger.WithError(err).Debug("Retrying again after
                        ↪  error detected")
191                     // Discard error because we will retry manually
192                     err = nil
193             }
194             if err == nil {
195                     err = manager.Delete(name)
196             }
197     }
198     for stateIndex < len(states) && !shouldDispatch && err == nil {
199             shouldDispatch = index <
                ↪  len(states[stateIndex].Habilitations)
200             states[stateIndex].Habilitations =
                ↪  states[stateIndex].Habilitations[index:]
201             if !shouldDispatch {
202                     stateIndex++
203                     index = 0
```

```
204                 }
205         }
206         var result pipeline.FutureID
207         var results []pipeline.FutureID
208         if err == nil {
209                 if shouldDispatch {
210                         results, err =
                        ↪  pipe.Dispatch(b.options.Tasks.FetchHabilitations,
                        ↪  states)
211                 } else {
212                         var names []string
213                         hasher := pool.GetSha1()
214                         for _, stateObj := range states {
215                                 names = append(names, stateObj.Names...)
216                                 if err == nil {
217                                         _, err = io.WriteString(hasher,
                                        ↪  stateObj.Prefix)
218                                 }
219                         }
220                         files := make([]interface{}, 0, len(names)+1)
221                         now := b.clk.Now()
222                         files = append(files,
                        ↪  fmt.Sprintf("%x-%d-%d.partial_2",
                        ↪  hasher.Sum(nil), now.Unix(),
                        ↪  now.Nanosecond()))
223                         pool.PutSha1(hasher)
224                         for _, name := range names {
225                                 files = append(files, name)
226                         }
227                         if err == nil {
228                                 results, err =
                                ↪  pipe.Dispatch(b.options.Tasks.JoinFiles,
                                ↪  files...)
229                         }
230                 }
231         }
232         if len(results) > 0 {
233                 result = results[0]
234         }
```

```
235         return result, err
236 }
237
238 // Bot defines the basic interface to start the processing of the data
239 type Bot interface {
240         Start(pipe pipeline.Pipeline, target string) (pipeline.FutureID,
        ↪  error)
241 }
242
243 // NewBot Gets a new Bot
244 func NewBot(disp pipeline.Dispatcher, clk clock.Clock, handler Handler,
    ↪  opt PipelineOptions) (Bot, error) {
245         opt = getPipelineDefaults(opt)
246         instance := &bot{
247                 options: opt,
248                 handler: handler,
249                 clk:     clk,
250         }
251         err := disp.Register(opt.Tasks.FetchHabilitations,
        ↪  instance.fetchHabilitations)
252         if err == nil {
253                 err = disp.Register(opt.Tasks.Start, instance.start)
254         }
255         if err == nil {
256                 err = disp.Register(opt.Tasks.FetchCampi,
                ↪  instance.fetchCampi)
257         }
258         if err == nil {
259                 err = disp.Register(opt.Tasks.BalanceHabilitations,
                ↪  instance.balanceHabilitations)
260         }
261         if err == nil {
262                 err = disp.Register(opt.Tasks.JoinFiles,
                ↪  instance.joinFiles)
263         }
264         return instance, err
265 }
```

Arquivo credit.go

```go
package usp2habilitations

func classCreditToHour(credit int) int {
        return credit * 15
}

func workCreditToHour(credit int) int {
        return credit * 30
}
```

Arquivo errors.go

```go
package usp2habilitations

import (
        "fmt"

        "github.com/fjorgemota/gurudamatricula/robot/format"
)

type errUnexpectedValue struct {
        value string
}

func (err errUnexpectedValue) Error() string {
        return fmt.Sprintf("semantic: Unexpected value '%s'", err.value)
}

func newErrUnexpectedValue(value string) error {
        return errUnexpectedValue{
                value: value,
        }
}

type errUnexpectedRelatedDiscipline struct {
        value string
}

func (err errUnexpectedRelatedDiscipline) Error() string {
```

```
28          return fmt.Sprintf("semantic: Unexpected related discipline type
            ↪  '%s'", err.value)
29  }
30
31  func newErrUnexpectedRelatedDiscipline(value string) error {
32          return errUnexpectedRelatedDiscipline{
33                  value: value,
34          }
35  }
36
37  type errOnString struct {
38          base  error
39          value string
40  }
41
42  func (err errOnString) Cause() error {
43          return err.base
44  }
45
46  func (err errOnString) Error() string {
47          return fmt.Sprintf("%s on string '%s'", err.base.Error(),
            ↪  err.value)
48  }
49
50  func newErrOnString(base error, str string) error {
51          return errOnString{
52                  base:  base,
53                  value: str,
54          }
55  }
56
57  type errTable struct {
58          base  error
59          elem  string
60          index int
61  }
62
63  func (err errTable) Cause() error {
64          return err.base
```

```
65  }
66
67  func (err errTable) Error() string {
68          return fmt.Sprintf("%s on %s %d", err.base.Error(), err.elem,
            ↪  err.index)
69  }
70
71  func newErrColumn(base error, column int) error {
72          return errTable{
73                  base:  base,
74                  elem:  "column",
75                  index: column,
76          }
77  }
78
79  func newErrRow(base error, row int) error {
80          return errTable{
81                  base:  base,
82                  elem:  "row",
83                  index: row,
84          }
85  }
86
87  type errExpected struct {
88          base     error
89          expected string
90  }
91
92  func (err errExpected) Cause() error {
93          return err.base
94  }
95
96  func (err errExpected) Error() string {
97          return fmt.Sprintf("%s, expected '%s'", err.base.Error(),
            ↪  err.expected)
98  }
99
100 func newErrExpected(base error, expected string) error {
101         return errExpected{
```

```
102                 base:     base,
103                 expected: expected,
104         }
105 }
106
107 type errHabilitation struct {
108         base         error
109         habilitation *format.Habilitation
110 }
111
112 func (err errHabilitation) Cause() error {
113         return err.base
114 }
115
116 func (err errHabilitation) Error() string {
117         codhab := err.habilitation.ID[:len(err.habilitation.ID)-1]
118         tipo := string(err.habilitation.ID[len(err.habilitation.ID)-1])
119         codcg := err.habilitation.Course.ID[:2]
120         return fmt.Sprintf("%s, on habilitation %s - '%s', and course %s -
          ↪  '%s' -
          ↪  https://uspdigital.usp.br/jupiterweb/listarGradeCurricular?codcg=%s&codcur=%
          ↪  ", err.base.Error(), err.habilitation.ID,
          ↪  err.habilitation.Name, err.habilitation.Course.ID,
          ↪  err.habilitation.Course.Name, codcg,
          ↪  err.habilitation.Course.ID, codhab, tipo)
121 }
122
123 func newErrHabilitation(base error, hab *format.Habilitation) error {
124         return errHabilitation{
125                 base:         base,
126                 habilitation: hab,
127         }
128 }
```

Arquivo fetcher.go

```
1 package usp2habilitations
2
3 import (
4         "bytes"
```

```
5           "io"
6           "net/http"
7           "net/url"
8
9           "github.com/fjorgemota/gurudamatricula/robot/format"
10          "github.com/fjorgemota/gurudamatricula/robot/util/attrs"
11          "github.com/fjorgemota/gurudamatricula/robot/util/uspreq"
12          "github.com/fjorgemota/gurudamatricula/util/pool"
13
14          "golang.org/x/net/html"
15          "golang.org/x/net/html/atom"
16  )
17
18  const coursesIndex = "https://uspdigital.usp.br/jupiterweb/jupCursoLista"
19
20  const habilitationIndex =
    ↪  "https://uspdigital.usp.br/jupiterweb/listarGradeCurricular"
21
22  const disciplineIndex =
    ↪  "https://uspdigital.usp.br/jupiterweb/obterDisciplina"
23
24  func GetHabililitations(client *http.Client, stringCache pool.StringCache,
    ↪  campus format.Campus, ID, userAgent string) ([]format.Habilitation,
    ↪  error) {
25          var habs []format.Habilitation
26          req, err := uspreq.GetRequest(coursesIndex, url.Values{
27                  "tipo":  []string{"N"},
28                  "codcg": []string{campus.ID},
29          }, userAgent)
30          var resp *http.Response
31          if err == nil {
32                  resp, err = client.Do(req)
33          }
34          var reader io.Reader
35          if err == nil {
36                  reader, err = uspreq.GetDoc(req, resp, ID)
37          }
38          if err == nil {
39                  doc := html.NewTokenizer(reader)
```

```
40                tableIndex := -1
41                columnIndex := -1
42                columnCount := -1
43                tmp := pool.GetBytesBuffer()
44                attrTmp := pool.GetBytesBuffer()
45                defer pool.PutBytesBuffer(tmp)
46                defer pool.PutBytesBuffer(attrTmp)
47                var hab format.Habilitation
48                setHabilitationTables(&hab)
49                setHabilitationLimits(&hab)
50                for err == nil {
51                        tagType := doc.Next()
52                        if tagType == html.ErrorToken {
53                                err = doc.Err()
54                        } else if tagType == html.StartTagToken {
55                                tag, hasNextAttr := doc.TagName()
56                                tagName := atom.String(tag)
57                                switch tagName {
58                                case "table":
59                                        tableIndex++
60                                        columnIndex = 0
61                                case "tr":
62                                        columnIndex = 0
63                                case "td":
64                                        columnIndex++
65                                case "a":
66                                        if columnIndex == 1 {
67                                                var attr []byte
68                                                attr, err =
                                                ↪  attrs.GetAttrBytes(doc,
                                                ↪  "href", hasNextAttr,
                                                ↪  attrTmp)
69                                                if err == nil {
70                                                        path :=
                                                        ↪  stringCache.String(attr)
71                                                        var u *url.URL
72                                                        u, err =
                                                        ↪  url.Parse(path)
```

```
73                                                  if err == nil &&
                                                    ↪  u.Path ==
                                                    ↪  "listarGradeCurricular"
                                                    ↪  {
74                                                      q :=
                                                        ↪  u.Query()
75                                                      hab =
                                                        ↪  format.Habilita
76                                                      hab.Course.ID
                                                        ↪  =
                                                        ↪  q.Get("codcur")
77                                                      hab.ID =
                                                        ↪  q.Get("codhab")
                                                        ↪  +
                                                        ↪  q.Get("tipo")
78                                                      err =
                                                        ↪  newErrHabilitat
                                                        ↪  " on
                                                        ↪  type"),
                                                        ↪  &hab)
79                                                      if
                                                        ↪  len(q.Get("tipo
                                                        ↪  == 1
                                                        ↪  {
80                                                          err
                                                            ↪  =
                                                            ↪  nil
81                                                      }
82                                                      setHabilitationTabl
83                                                      setHabilitationLimi
84                                                  }
85                                              }
86                                          }
87                                  }
88                          } else if tagType == html.EndTagToken {
89                                  tag, _ := doc.TagName()
90                                  tagName := atom.String(tag)
91                                  switch tagName {
92                                  case "table":
```

```
93                                      columnIndex = -1
94                                      columnCount = -1
95                                  case "tr":
96                                      if columnCount == 2 ||
                                        ↪  columnCount == 3 {
97                                          if hab.Course.ID != "" &&
                                            ↪  hab.ID != "" {
98                                              habs =
                                                ↪  append(habs,
                                                ↪  hab)
99                                              hab =
                                                ↪  format.Habilitation{}
100                                             setHabilitationTables(&hab)
101                                             setHabilitationLimits(&hab)
102                                         }
103                                     }
104                                     columnCount = columnIndex
105                                 case "td":
106                                     if hab.ID == "" || tmp.Len() == 0
                                        ↪  {
107                                         continue
108                                     }
109                                     str :=
                                        ↪  stringCache.String(tmp.Bytes())
110                                     tmp.Reset()
111                                     switch columnIndex {
112                                     case 1:
113                                         hab.Name = str
114                                         hab.Course.Name = str
115                                         if hab.Name ==
                                            ↪  hab.Course.ID+"
                                            ↪  "+hab.ID[:len(hab.ID)-1]
                                            ↪  {
116                                             hab.Course.Name =
                                                ↪  ""
117                                             hab.Name = ""
118                                         }
119                                     case 2:
120                                         if hab.Name == "" {
```

```
121                                                     hab.Name = str
122                                                     tmp.Reset()
123                                                     continue
124                                             }
125                                             fallthrough
126                                     case 3:
127                                             hab.Tables[0].Rows =
                                                ↪ append(hab.Tables[0].Rows,
                                                ↪ map[string]string{"key":
                                                ↪ "Período", "value":
                                                ↪ str})
128                                     }
129                             }
130                     } else if tagType == html.TextToken && hab.ID !=
                        ↪ "" {
131                             tmp.Write(bytes.TrimSpace(doc.Text()))
132                     }
133             }
134             if err == io.EOF {
135                     err = nil
136             }
137     }
138     return habs, err
139 }
140
141 func GetOldHabililitations(client *http.Client, stringCache
    ↪ pool.StringCache, campus format.Campus, ID, userAgent string)
    ↪ ([]format.Habilitation, error) {
142     var habs []format.Habilitation
143     req, err := uspreq.GetRequest(coursesIndex, url.Values{
144             "tipo":  []string{"V"},
145             "codcg": []string{campus.ID},
146     }, userAgent)
147     var resp *http.Response
148     if err == nil {
149             resp, err = client.Do(req)
150     }
151     var reader io.Reader
152     if err == nil {
```

```
153                 reader, err = uspreq.GetDoc(req, resp, ID)
154         }
155         if err == nil {
156                 doc := html.NewTokenizer(reader)
157                 tableIndex := -1
158                 columnIndex := -1
159                 columnCount := -1
160                 tmp := pool.GetBytesBuffer()
161                 attrTmp := pool.GetBytesBuffer()
162                 defer pool.PutBytesBuffer(tmp)
163                 defer pool.PutBytesBuffer(attrTmp)
164                 var hab format.Habilitation
165                 setHabilitationTables(&hab)
166                 setHabilitationLimits(&hab)
167                 for err == nil {
168                         tagType := doc.Next()
169                         if tagType == html.ErrorToken {
170                                 err = doc.Err()
171                         } else if tagType == html.StartTagToken {
172                                 tag, hasNextAttr := doc.TagName()
173                                 tagName := atom.String(tag)
174                                 switch tagName {
175                                 case "table":
176                                         tableIndex++
177                                         columnIndex = 0
178                                 case "tr":
179                                         columnIndex = 0
180                                 case "td":
181                                         columnIndex++
182                                 case "a":
183                                         if columnIndex == 1 {
184                                                 var attr []byte
185                                                 attr, err =
                                                    ↪ attrs.GetAttrBytes(doc,
                                                    ↪ "href", hasNextAttr,
                                                    ↪ attrTmp)
186                                                 if err == nil {
187                                                         path :=
                                                            ↪ stringCache.String(attr)
```

```
188                                                     var u *url.URL
189                                                     u, err =
                                                        ↪  url.Parse(path)
190                                                     if err == nil &&
                                                        ↪  u.Path ==
                                                        ↪  "listarGradeCurricular"
                                                        ↪  {
191                                                         q :=
                                                            ↪  u.Query()
192                                                         hab =
                                                            ↪  format.Habilita
193                                                         hab.Course.ID
                                                            ↪  =
                                                            ↪  q.Get("codcur")
194                                                         hab.ID =
                                                            ↪  q.Get("codhab")
                                                            ↪  +
                                                            ↪  q.Get("tipo")
195                                                         err =
                                                            ↪  newErrHabilitat
                                                            ↪  " on
                                                            ↪  type"),
                                                            ↪  &hab)
196                                                         if
                                                            ↪  len(q.Get("tipo
                                                            ↪  == 1
                                                            ↪  {
197                                                             err
                                                                ↪  =
                                                                ↪  nil
198                                                         }
199                                                         setHabilitationTabl
200                                                         setHabilitationLimi
201                                                     }
202                                                 }
203                                             }
204                                         }
205                                 } else if tagType == html.EndTagToken {
206                                         tag, _ := doc.TagName()
```

```
207                         tagName := atom.String(tag)
208                         switch tagName {
209                         case "table":
210                                 columnIndex = -1
211                                 columnCount = -1
212                         case "tr":
213                                 if columnCount == 4 {
214                                         if hab.Course.ID != "" &&
                                            ↪ hab.ID != "" {
215                                                 habs =
                                                    ↪ append(habs,
                                                    ↪ hab)
216                                                 hab =
                                                    ↪ format.Habilitation{}
217                                                 setHabilitationTables(&hab)
218                                                 setHabilitationLimits(&hab)
219                                         }
220                                 }
221                                 columnCount = columnIndex
222                         case "td":
223                                 if hab.ID == "" || tmp.Len() == 0
                                    ↪ {
224                                         continue
225                                 }
226                                 str :=
                                    ↪ stringCache.String(tmp.Bytes())
227                                 tmp.Reset()
228                                 switch columnIndex {
229                                 case 2:
230                                         hab.Course.Name = str
231                                 case 3:
232                                         hab.Name = str
233                                 case 4:
234                                         hab.Tables[0].Rows =
                                            ↪ append(hab.Tables[0].Rows,
                                            ↪ map[string]string{"key":
                                            ↪ "Período", "value":
                                            ↪ str})
235                                 }
```

```
236                                     }
237                             } else if tagType == html.TextToken && hab.ID !=
                                ↪  "" {
238                                     tmp.Write(bytes.TrimSpace(doc.Text()))
239                             }
240                     }
241                     if err == io.EOF {
242                             err = nil
243                     }
244             }
245             return habs, err
246     }
247
248     func fetchHabilitation(client *http.Client, campus format.Campus, hab
        ↪  format.Habilitation, ID, userAgent string) (io.Reader, error) {
249             req, err := uspreq.GetRequest(habilitationIndex, url.Values{
250                     "codcur": []string{hab.Course.ID},
251                     "codhab": []string{hab.ID[:len(hab.ID)-1]},
252                     "codcg":  []string{campus.ID},
253                     "tipo":   []string{string(hab.ID[len(hab.ID)-1])},
254             }, userAgent)
255             var resp *http.Response
256             if err == nil {
257                     resp, err = client.Do(req)
258             }
259             var reader io.Reader
260             if err == nil {
261                     reader, err = uspreq.GetDoc(req, resp, ID)
262             }
263             return reader, err
264     }
265
266     func fetchDiscipline(client *http.Client, discipline *format.Discipline,
        ↪  hab format.Habilitation, ID, userAgent string) (io.Reader, error) {
267             req, err := uspreq.GetRequest(disciplineIndex, url.Values{
268                     "sgldis": []string{discipline.Code},
269                     "codcur": []string{hab.Course.ID},
270                     "codhab": []string{hab.ID[:len(hab.ID)-1]},
271             }, userAgent)
```

```
272         var resp *http.Response
273         if err == nil {
274                 resp, err = client.Do(req)
275         }
276         var reader io.Reader
277         if err == nil {
278                 reader, err = uspreq.GetDoc(req, resp, ID)
279         }
280         return reader, err
281 }
```

Arquivo habilitation.go

```
1  package usp2habilitations
2
3  import (
4          "strings"
5
6          "github.com/fjorgemota/gurudamatricula/robot/format"
7  )
8
9  var habilitationWorks = []struct{ ID, Name string }{{"classes", "Aula"},
   ↪ {"work", "Trabalho"}}
10 var habilitationWorksTypes = []struct{ ID, Name string }{{"required",
   ↪  "Obrigatória"}, {"optional_free", "Optativa Livre"},
   ↪ {"optional_elective", "Optativa Eletiva"}}
11
12 func getWorkTypeID(disciplineType string) string {
13         var ID string
14         for _, workType := range habilitationWorksTypes {
15                 index := strings.LastIndex(workType.Name, " ")
16                 if index == -1 {
17                         index = 0
18                 }
19                 if strings.Contains(disciplineType,
                   ↪  workType.Name[index:]) {
20                         ID = workType.ID
21                         break
22                 }
23         }
```

```
24          return ID
25  }
26
27  func setHabilitationTables(hab *format.Habilitation) {
28          if len(hab.Tables) > 0 {
29                  return
30          }
31          var details format.Table
32          details.ID = "details"
33          details.Columns = []format.Column{{ID: "key", Name: ""}, {ID:
            ↪  "value", Name: ""}}
34          hab.Tables = append(hab.Tables, details)
35          var vacancies format.Table
36          vacancies.ID = "vacancies"
37          vacancies.Columns = []format.Column{{ID: "type", Name: "Tipo"}}
38          for _, work := range habilitationWorks {
39                  vacancies.Columns = append(vacancies.Columns,
                    ↪  format.Column{ID: work.ID, Name: work.Name})
40          }
41          vacancies.Columns = append(vacancies.Columns, format.Column{ID:
            ↪  "total", Name: "total"})
42          hab.Tables = append(hab.Tables, vacancies)
43  }
44
45  func setHabilitationLimits(hab *format.Habilitation) {
46          if len(hab.Limits) > 0 {
47                  return
48          }
49          var duration format.Limit
50          duration.ID = "duration"
51          duration.Context = "user"
52          duration.Name = "Duração"
53          duration.Unit = "-"
54          duration.Values = make(map[string]float64)
55          hab.Limits = append(hab.Limits, duration)
56          for _, work := range habilitationWorks {
57                  for _, typ := range habilitationWorksTypes {
58                          var workload format.Limit
```

```
59                          workload.ID = "workload_" + work.ID + "_" +
                            ↪  typ.ID
60                          workload.Context = "user"
61                          workload.Name = "Carga Horária - " + work.Name +
                            ↪  " - " + typ.Name
62                          workload.Unit = "horas"
63                          workload.Values = make(map[string]float64)
64                          hab.Limits = append(hab.Limits, workload)
65                  }
66          }
67  }
68
69  func cleanHabilitation(hab *format.Habilitation) {
70          limits := hab.Limits[:0]
71          for _, limit := range hab.Limits {
72                  if len(limit.Values) > 0 {
73                          limits = append(limits, limit)
74                  }
75          }
76          hab.Limits = limits
77          shouldClear := true
78          for _, row := range hab.Tables[0].Rows {
79                  if len(row["value"]) > 0 {
80                          shouldClear = false
81                          break
82                  }
83          }
84          if shouldClear {
85                  hab.Tables = hab.Tables[1:]
86          }
87          shouldClear = true
88          i := len(hab.Tables) - 1
89          for _, row := range hab.Tables[i].Rows {
90                  if (len(row["classes"]) > 0 && row["classes"] != "0") ||
                    ↪  (len(row["work"]) > 0 && row["work"] != "0") {
91                          shouldClear = false
92                          break
93                  }
94          }
```

```
95          if shouldClear {
96                  hab.Tables = hab.Tables[:i]
97          }
98  }
```

Arquivo options.go

```
1   package usp2habilitations
2
3   import (
4           "runtime"
5           "time"
6   )
7
8   type Tasks struct {
9           Start                string
10          FetchCampi           string
11          BalanceHabilitations string
12          FetchHabilitations   string
13          JoinFiles            string
14  }
15
16  // BaseOptions define base options for the public API
17  type BaseOptions struct {
18          DelayBetweenRequests time.Duration
19          UserAgent            string
20  }
21
22  // HabilitationOptions define some options for the public API
23  type HabilitationOptions struct {
24          Base           BaseOptions
25          GetDisciplines bool
26  }
27
28  // Pipeline defines options that are needed for the Pipeline API
29  type PipelineOptions struct {
30          Crawler          HabilitationOptions
31          Tasks            Tasks
32          BatchSize        int
33          ParallelRequests int
```

```
34          MaxRetries       int
35  }
36
37  func getBaseDefaults(opt BaseOptions) BaseOptions {
38          if opt.DelayBetweenRequests <= 0 {
39                  opt.DelayBetweenRequests = 2 * time.Second
40          }
41          if opt.UserAgent == "" {
42                  opt.UserAgent = "Mozilla/5.0 (X11; Linux x86_64)
                    ↪ AppleWebKit/537.36 (KHTML, like Gecko)
                    ↪ Chrome/60.0.3112.101 Safari/537.36"
43          }
44          return opt
45  }
46
47  func getPipelineDefaults(opt PipelineOptions) PipelineOptions {
48          if opt.Tasks.FetchHabilitations == "" {
49                  opt.Tasks.FetchHabilitations =
                    ↪ "usp.semantic.fetch_habilitations"
50          }
51          if opt.Tasks.FetchCampi == "" {
52                  opt.Tasks.FetchCampi = "usp.semantic.fetch_campi"
53          }
54          if opt.Tasks.JoinFiles == "" {
55                  opt.Tasks.JoinFiles = "usp.semantic.join"
56          }
57          if opt.Tasks.BalanceHabilitations == "" {
58                  opt.Tasks.BalanceHabilitations = "usp.semantic.balance"
59          }
60          if opt.Tasks.Start == "" {
61                  opt.Tasks.Start = "usp.semantic.start"
62          }
63          if opt.BatchSize <= 0 {
64                  opt.BatchSize = 10
65          }
66          if opt.ParallelRequests <= 0 {
67                  opt.ParallelRequests = runtime.NumCPU()
68          }
69          if opt.MaxRetries == 0 {
```

```
70                  opt.MaxRetries = 3
71          }
72          if opt.MaxRetries < 0 {
73                  opt.MaxRetries = 0
74          }
75          opt.Crawler.Base = getBaseDefaults(opt.Crawler.Base)
76          return opt
77  }
```

Arquivo parser.go

```
1   package usp2habilitations
2
3   import (
4           "bytes"
5           "crypto/sha1"
6           "encoding/hex"
7           "io"
8           "strconv"
9           "strings"
10          "unicode/utf8"
11
12          "github.com/fjorgemota/gurudamatricula/robot/format"
13          "github.com/fjorgemota/gurudamatricula/robot/util/attrs"
14          "github.com/fjorgemota/gurudamatricula/util/pool"
15          "github.com/gofrs/uuid"
16          "golang.org/x/net/html"
17          "golang.org/x/net/html/atom"
18  )
19
20  func stripSpaces(bs []byte) string {
21          words := bytes.Split(bs, []byte(" "))
22          result := make([]byte, 0, len(bs))
23          for _, word := range words {
24                  word = bytes.TrimSpace(word)
25                  if len(word) > 0 {
26                          if len(result) > 0 {
27                                  result = append(result, ' ')
28                          }
29                          result = append(result, word...)
```

```
30                  }
31          }
32          return string(result)
33  }
34
35  func hash(bs []byte) string {
36          result := sha1.Sum(bs)
37          return hex.EncodeToString(result[:])
38  }
39
40  func fixRune(r rune) rune {
41          if r == utf8.RuneError {
42                  return -1
43          }
44          return r
45  }
46
47  func fixEncoding(s string) string {
48          return strings.Map(fixRune, s)
49  }
50
51  func parseDisciplinePage(discipline *format.Discipline, hab
    ↪  format.Habilitation, reader io.Reader, intCache pool.AtoiCache,
    ↪  stringCache pool.StringCache) error {
52          var err error
53          state := -1
54          doc := html.NewTokenizer(reader)
55          tmp := pool.GetBytesBuffer()
56          attrTmp := pool.GetBytesBuffer()
57          defer pool.PutBytesBuffer(tmp)
58          defer pool.PutBytesBuffer(attrTmp)
59          var fieldName string
60          table := format.Table{
61                  ID:    "details",
62                  Title: "Detalhes",
63                  Columns: []format.Column{
64                          format.Column{
65                                  ID:   "key",
66                                  Name: "",
```

```
67                          },
68                          format.Column{
69                                  ID:    "value",
70                                  Name: "",
71                          },
72                  },
73          }
74          for err == nil {
75                  tagType := doc.Next()
76                  if tagType == html.ErrorToken {
77                          err = doc.Err()
78                  } else if tagType == html.StartTagToken {
79                          tag, hasNextAttr := doc.TagName()
80                          tagName := atom.String(tag)
81                          switch tagName {
82                          case "table":
83                                  state = 0
84                          case "tr":
85                                  fallthrough
86                          case "br":
87                                  if tmp.Len() > 0 {
88                                          err = tmp.WriteByte('\n')
89                                  }
90                          case "td":
91                                  var value []byte
92                                  value, err = attrs.GetAttrBytes(doc,
                                    ↪  "align", hasNextAttr, attrTmp)
93                                  if bytes.Equal(value, []byte("CENTER")) {
94                                          state = -2
95                                  } else {
96                                          state = 2
97                                  }
98                          case "b":
99                                  if len(fieldName) > 0 && tmp.Len() > 0 {
100                                         value :=
                                           ↪  strings.TrimSpace(stringCache.String(tm
101                                         table.Rows = append(table.Rows,
                                           ↪  map[string]string{
102                                                 "key":    fieldName,
```

```
                                        "value": value,
                                })
                                tmp.Reset()
                        }
                        if state == 2 {
                                state = 1
                        }
                case "a":
                        state = -3
                case "i":
                        state = -4
                }
        } else if tagType == html.EndTagToken {
                tag, _ := doc.TagName()
                tagName := atom.String(tag)
                switch {
                case state >= 0 && tagName == "table":
                        state = -1
                case state == 1:
                        fieldName =
                        ↪  strings.TrimSuffix(stringCache.String(tmp.Bytes()),
                        ↪  ":")
                        state = 0
                        tmp.Reset()
                case state == 2 && len(fieldName) > 0:
                        state = 0
                }
        } else if tagType == html.TextToken && state > -1 {
                tmp.Write(bytes.TrimSpace(doc.Text()))
        }
}
if err == io.EOF {
        err = nil
}
if err == nil && len(fieldName) > 0 && tmp.Len() > 0 {
        value :=
        ↪  strings.TrimSpace(stringCache.String(tmp.Bytes()))
        table.Rows = append(table.Rows, map[string]string{
                "key":   fieldName,
```

```
                        "value": value,
                })
        }
        if err == nil && len(table.Rows) > 0 {
                for _, row := range table.Rows {
                        for key, value := range row {
                                row[key] = fixEncoding(value)
                        }
                        if row["key"] == "Programa Resumido" {
                                discipline.Description = row["value"]
                        }
                }
                discipline.Tables = append(discipline.Tables, table)
        }
        return err
}

func parseHabilitationPage(hab format.Habilitation, reader io.Reader,
    ↪ intCache pool.AtoiCache, stringCache pool.StringCache)
    ↪ (format.Habilitation, error) {
        doc := html.NewTokenizer(reader)
        columnIndex := -1
        infoType := -1
        rowIndex := -1
        tableIndex := -1
        columnCount := -1
        var disciplineType, disciplineTypeID string
        randomPlaceholderHandle, err := uuid.NewV4()
        randomPlaceholder := "OR-"
        if err == nil {
                randomPlaceholder += randomPlaceholderHandle.String()
        }
        var habName, courseName string
        tmp := pool.GetBytesBuffer()
        attrTmp := pool.GetBytesBuffer()
        defer pool.PutBytesBuffer(tmp)
        defer pool.PutBytesBuffer(attrTmp)
        idByCode := make(map[string]string)
        setHabilitationTables(&hab)
```

```
176         setHabilitationLimits(&hab)
177         for err == nil {
178                 tagType := doc.Next()
179                 if tagType == html.ErrorToken {
180                         err = doc.Err()
181                 } else if tagType == html.StartTagToken {
182                         tag, hasNextAttr := doc.TagName()
183                         tagName := atom.String(tag)
184                         switch tagName {
185                         case "table":
186                                 columnIndex = 0
187                                 rowIndex = 0
188                                 tableIndex++
189                         case "tr":
190                                 rowIndex++
191                                 var value []byte
192                                 value, err = attrs.GetAttrBytes(doc,
                                    ↪ "bgcolor", hasNextAttr, attrTmp)
193                                 if bytes.Equal(value, []byte("#658CCF"))
                                    ↪ {
194                                         // Can use to detect tables of
                                            ↪ disciplines
195                                         infoType = 5
196                                 } else if bytes.Equal(value,
                                    ↪ []byte("#FFFFFF")) && (((columnCount
                                    ↪ == 7 || columnCount == 8) &&
                                    ↪ (infoType == 7 || infoType == 6 ||
                                    ↪ infoType == 9)) || (infoType == 8 &&
                                    ↪ columnCount == 2)) {
197                                         // Can use to detect disciplines
198                                         infoType = 7
199                                 } else if bytes.Equal(value,
                                    ↪ []byte("#CCCCCC")) {
200                                         // Can use to detect steps
201                                         infoType = 6
202                                 } else if len(value) == 0 && infoType ==
                                    ↪ 7 {
203                                         infoType = 9
204                                 }
```

```
205                         case "td":
206                                 columnIndex++
207                                 var value []byte
208                                 value, err = attrs.GetAttrBytes(doc,
                                    ↪ "colspan", hasNextAttr, attrTmp)
209                                 if err == nil && bytes.Equal(value,
                                    ↪ []byte("2")) && (infoType == 7 ||
                                    ↪ infoType == 8 || infoType == 9) &&
                                    ↪ columnIndex == 1 {
210                                         infoType = -1
211                                 } else if err == nil && bytes.Equal(value,
                                    ↪ []byte("4")) && columnIndex == 1 &&
                                    ↪ len(hab.Steps) > 0 {
212                                         infoType = 8
213                                 }
214                         }
215                 } else if tagType == html.EndTagToken {
216                         tag, _ := doc.TagName()
217                         tagName := atom.String(tag)
218                         switch tagName {
219                         case "table":
220                                 columnIndex = -1
221                                 infoType = -1
222                                 rowIndex = -1
223                                 columnCount = -1
224                         case "tr":
225                                 columnCount = columnIndex
226                                 columnIndex = 0
227                         case "td":
228                                 switch infoType {
229                                 case -1:
230                                         if rowIndex < 3 && tableIndex > 2
                                            ↪ {
231                                                 if columnIndex == 1 {
232                                                         bs := tmp.Bytes()
233                                                         index :=
                                                            ↪ bytes.IndexByte(bs,
                                                            ↪ ':')
234                                                         if index > -1 {
```

```
key := stringCache.String(b
value := stringCache.String(b
if key == "Curso" {
    if courseName == "" {
        courseNa = valu
    }
    habName = value
    break
}
if len(value) > 0 {
    hab.Tables[0].Ro = append(hab.T
    map[string]s
    key,
    "value":
    value})
} else {
    hab.Tables[0].Ro = append(hab.T
    map[string]s
    key})
```

```
248                                                                 infoType
                                                                    ↪  =
                                                                    ↪  1
249                                                             }
250                                                         } else if
                                                            ↪  stringCache.Transform(t
                                                            ↪  stripSpaces)
                                                            ↪  == "Carga
                                                            ↪  Horária" {
251                                                             infoType
                                                                ↪  = 4
252                                                         }
253                                                 } else if
                                                    ↪  bytes.Contains(tmp.Bytes(),
                                                    ↪  []byte("Duração")) {
254                                                     infoType = 2
255                                                 }
256                                         }
257                                 case 1:
258                                         l := len(hab.Tables[0].Rows) - 1
259                                         hab.Tables[0].Rows[l]["value"] =
                                            ↪  stringCache.String(bytes.TrimSpace(tmp.
260                                         if hab.Tables[0].Rows[l]["key"]
                                            ↪  == "Data de Início" {
261                                             hab.Course.Version =
                                                ↪  hab.Tables[0].Rows[l]["value"]
262                                         }
263                                         infoType = -1
264                                 case 2:
265                                         bs := tmp.Bytes()
266                                         if len(bs) == 0 {
267                                             continue
268                                         }
269                                         infoType = 3
270                                         if bytes.Contains(bs,
                                            ↪  []byte("Mínima")) {
271                                             hab.Limits[0].Values["min"]
                                                ↪  = -1
```

```
272                             } else if bytes.Contains(bs,
                                ↪  []byte("Máxima")) {
273                                     hab.Limits[0].Values["max"]
                                        ↪  = -1
274                             } else if bytes.Contains(bs,
                                ↪  []byte("Ideal")) {
275                                     hab.Limits[0].Values["ideal"]
                                        ↪  = -1
276                             }
277                     case 3:
278                             infoType = 2
279                             bs := tmp.Bytes()
280                             bs = bytes.TrimSpace(bs)
281                             values := bytes.Split(bs,
                                ↪  []byte(" "))
282                             if len(values) == 2 {
283                                     unit :=
                                        ↪  stringCache.String(values[1])
284                                     if hab.Limits[0].Unit ==
                                        ↪  "-" {
285                                             hab.Limits[0].Unit
                                                ↪  = unit
286                                     }
287                                     if unit ==
                                        ↪  hab.Limits[0].Unit {
288                                             var number int
289                                             number, err =
                                                ↪  intCache.Atoi(values[0])
290                                             for key, value :=
                                                ↪  range
                                                ↪  hab.Limits[0].Values
                                                ↪  {
291                                                     if value
                                                        ↪  == -1
                                                        ↪  &&
                                                        ↪  err
                                                        ↪  ==
                                                        ↪  nil {
```

```
292                                                             hab.Limits[
                                                                ↪  =
                                                                ↪  float64
293                                                             break
294                                                     }
295                                             }
296                                     } else {
297                                             err =
                                                ↪  newErrOnString(newErrUn
                                                ↪  string(bs))
298                                     }
299                             } else if len(values) > 2 {
300                                     // On some cases there
                                        ↪  are a unit without a
                                        ↪  value before.
301                                     // On these cases no
                                        ↪  error is emitted
302                                     err =
                                        ↪  newErrUnexpectedValue(string(bs
303                             } else {
304                                     for key, value := range
                                        ↪  hab.Limits[0].Values
                                        ↪  {
305                                             if value == -1 &&
                                                ↪  err == nil {
306                                                     delete(hab.Limits[0
                                                        ↪  key)
307                                             }
308                                     }
309                             }
310                     case 4:
311                             if rowIndex == 1 {
312                                     header :=
                                        ↪  stringCache.Transform(tmp.Bytes
                                        ↪  stripSpaces)
313                                     var expected string
314                                     if err == nil {
315                                             switch
                                                ↪  columnIndex {
```

```
316                                         case 2, 3:
317                                                 expected
                                                    ↪  =
                                                    ↪  habilitationWorks[co
318                                         case 4:
319                                                 expected
                                                    ↪  =
                                                    ↪  "Subtotal"
320                                         default:
321                                                 err =
                                                    ↪  newErrColumn(newErrU
                                                    ↪  columnIndex)
322                                         }
323                                 }
324                                 if err == nil && expected
                                    ↪  != header {
325                                         err =
                                            ↪  newErrExpected(newErrColumn(
                                            ↪  columnIndex),
                                            ↪  expected)
326                                 }
327                         } else {
328                                 if columnIndex == 1 {
329                                         typ :=
                                            ↪  stringCache.Transform(tmp.By
                                            ↪  stripSpaces)
330                                         var expected
                                            ↪  string
331                                         if err == nil {
332                                                 hab.Tables[1].Rows
                                                    ↪  =
                                                    ↪  append(hab.Tables[1]
                                                    ↪  make(map[string]stri
333                                                 row :=
                                                    ↪  len(hab.Tables[1].Ro
                                                    ↪  - 1
334                                                 hab.Tables[1].Rows[row][
                                                    ↪  = typ
```

```
335                                                 switch
                                                    ↪ rowIndex
                                                    ↪ {
336                                                 case 2, 3,
                                                    ↪ 4:
337                                                         expected
                                                            ↪ =
                                                            ↪ habilit
338                                                 case 5:
339                                                         expected
                                                            ↪ =
                                                            ↪ "Total"
340                                                 default:
341                                                         err
                                                            ↪ =
                                                            ↪ newErrR
                                                            ↪ rowInde
342                                                 }
343                                             }
344                                             if err == nil &&
                                                ↪ expected !=
                                                ↪ typ {
345                                                     err =
                                                        ↪ newErrExpected(
                                                        ↪ rowIndex),
                                                        ↪ expected)
346                                             }
347                                         } else if rowIndex <= 4
                                            ↪ && columnIndex <= 3
                                            ↪ && err == nil {
348                                             work :=
                                                ↪ habilitationWorks[colum
349                                             row :=
                                                ↪ len(hab.Tables[1].Rows)
                                                ↪ - 1
350                                             bs := tmp.Bytes()
351                                             if len(bs) > 0 {
```

```
352                                                 hab.Tables[1].Rows[row]
                                                    ↪  =
                                                    ↪  stringCache.String(b
353                                                 index :=
                                                    ↪  1 +
                                                    ↪  ((columnIndex
                                                    ↪  - 2) *
                                                    ↪  len(habilitationWork
                                                    ↪  +
                                                    ↪  (rowIndex
                                                    ↪  - 2)
354                                                 var value
                                                    ↪  int
355                                                 value,
                                                    ↪  err =
                                                    ↪  intCache.Atoi(bs)
356                                                 if err ==
                                                    ↪  nil
                                                    ↪  &&
                                                    ↪  tmp.Len()
                                                    ↪  > 0 {
357                                                     hab.Limits[index
                                                        ↪  =
                                                        ↪  float64(valu
358                                                 }
359                                         }
360                                 } else {
361                                         row :=
                                            ↪  len(hab.Tables[1].Rows)
                                            ↪  - 1
362                                         if columnIndex <=
                                            ↪  3 {
363                                             work :=
                                                ↪  habilitationWorks[co
364                                             hab.Tables[1].Rows[row]
                                                ↪  =
                                                ↪  stringCache.String(t
365                                         } else {
```

```
366                                                                     if
                                                                        ↪ hab.Tables[1].R
                                                                        ↪ != ""
                                                                        ↪ {
367                                                                         hab.Tables[
                                                                            ↪ +=
                                                                            ↪ "
                                                                            ↪ "
368                                                                     }
369                                                                     hab.Tables[1].Rows[
                                                                        ↪ +=
                                                                        ↪ stringCache.Tra
                                                                        ↪ stripSpaces)
370                                                             }
371                                                     }
372                                             }
373                                     case 5:
374                                             disciplineType =
                                                ↪ stringCache.Transform(tmp.Bytes(),
                                                ↪ stripSpaces)
375                                             disciplineTypeID =
                                                ↪ getWorkTypeID(disciplineType)
376                                     case 6:
377                                             var expected string
378                                             value :=
                                                ↪ stringCache.Transform(tmp.Bytes(),
                                                ↪ stripSpaces)
379                                             if err == nil {
380                                                     switch columnIndex {
381                                                     case 1:
382                                                             stepName :=
                                                                ↪ disciplineType
                                                                ↪ + " - " +
                                                                ↪ value
383                                                             hab.Steps =
                                                                ↪ append(hab.Steps,
                                                                ↪ format.Step{
384                                                                     ID:
                                                                        ↪ hash([]byte(ste
```

```
385                                                     Name:
                                                        ↪  stepName,
386                                                 })
387                                                 expected = value
388                                         case 2:
389                                                 expected =
                                                    ↪  "Créd.Aula"
390                                         case 3:
391                                                 expected =
                                                    ↪  "Créd.Trab."
392                                         case 4:
393                                                 expected = "CH"
394                                         case 5:
395                                                 expected = "CE"
396                                         case 6:
397                                                 expected = "CP"
398                                         case 7:
399                                                 expected = "ATPA"
400                                         default:
401                                                 err =
                                                    ↪  newErrColumn(newErrUnexpecte
                                                    ↪  columnIndex)
402                                         }
403                                 }
404                                 if err == nil && expected !=
                                    ↪  value {
405                                         err =
                                            ↪  newErrExpected(newErrColumn(newErrUn
                                            ↪  columnIndex),
                                            ↪  expected)
406                                 }
407                         case 7:
408                                 // Reads disciplines
409                                 l := len(hab.Steps) - 1
410                                 d :=
                                    ↪  len(hab.Steps[l].Disciplines)
                                    ↪  - 1
411                                 switch columnIndex {
412                                 case 1:
```

```
413                                 value :=
                                    ↪ stringCache.Transform(tmp.Bytes
                                    ↪ stripSpaces)
414                                 discipline :=
                                    ↪ format.Discipline{
415                                     ID:
                                        ↪ hash([]byte(hab.Course.
                                        ↪ + "|" +
                                        ↪ hab.ID + "|"
                                        ↪ +
                                        ↪ strconv.Itoa(l)
                                        ↪ + "|" +
                                        ↪ strconv.Itoa(d+1)
                                        ↪ + "|" +
                                        ↪ value)),
416                                     Code:      value,
417                                     GenericID:
                                        ↪ hash([]byte(value)),
418                                     Type:
                                        ↪ disciplineType,
419                                 }
420                                 idByCode[value] =
                                    ↪ discipline.ID
421                                 hab.Steps[l].Disciplines
                                    ↪ =
                                    ↪ append(hab.Steps[l].Disciplines
                                    ↪ discipline)
422                             case 2:
423                                 value :=
                                    ↪ stringCache.Transform(tmp.Bytes
                                    ↪ stripSpaces)
424                                 hab.Steps[l].Disciplines[d].Name
                                    ↪ = value
425                             case 3:
426                                 bs := tmp.Bytes()
427                                 var value int
428                                 value, err =
                                    ↪ intCache.Atoi(bs)
```

```
429                                 if err == nil && value >
                                    ↪  0 {
430                                         hab.Steps[l].Disciplines[d].Work
                                            ↪  =
                                            ↪  append(hab.Steps[l].Discipli
                                            ↪  format.DisciplineWorkload{
431                                                 ID:
                                                    ↪  "credits_classes",
432                                                 Name:
                                                    ↪  "Créditos
                                                    ↪  -
                                                    ↪  Aula",
433                                                 Unit:
                                                    ↪  "créditos",
434                                                 Value:
                                                    ↪  float64(value),
435                                         })
436                                         hab.Steps[l].Disciplines[d].Work
                                            ↪  =
                                            ↪  append(hab.Steps[l].Discipli
                                            ↪  format.DisciplineWorkload{
437                                                 ID:
                                                    ↪  "workload_"
                                                    ↪  +
                                                    ↪  habilitationWorks[0]
                                                    ↪  + "_"
                                                    ↪  +
                                                    ↪  disciplineTypeID,
438                                                 Name:
                                                    ↪  "Carga
                                                    ↪  Horária
                                                    ↪  - " +
                                                    ↪  habilitationWorks[0]
439                                                 Unit:
                                                    ↪  "horas",
440                                                 Value:
                                                    ↪  float64(classCreditT
441                                         })
442                                 }
```

```
443                                 case 4:
444                                     bs := tmp.Bytes()
445                                     var value int
446                                     value, err =
                                        ↪  intCache.Atoi(bs)
447                                     if err == nil && value >
                                        ↪  0 {
448                                         hab.Steps[l].Disciplines[d]
                                            ↪  =
                                            ↪  append(hab.Steps[l].Dis
                                            ↪  format.DisciplineWorklo
449                                             ID:
                                                ↪  "credits_work",
450                                             Name:
                                                ↪  "Créditos
                                                ↪  -
                                                ↪  Trabalho",
451                                             Unit:
                                                ↪  "créditos",
452                                             Value:
                                                ↪  float64(value),
453                                         })
454                                         hab.Steps[l].Disciplines[d]
                                            ↪  =
                                            ↪  append(hab.Steps[l].Dis
                                            ↪  format.DisciplineWorklo
455                                             ID:
                                                ↪  "workload_"
                                                ↪  +
                                                ↪  habilitationWor
                                                ↪  + "_"
                                                ↪  +
                                                ↪  disciplineTypeI
456                                             Name:
                                                ↪  "Carga
                                                ↪  Horária
                                                ↪  - " +
                                                ↪  habilitationWor
```

```
457                                                     Unit:
                                                        ↪  "horas",
458                                                     Value:
                                                        ↪  float64(workCreditTo
459                                                 })
460                                         }
461                                 case 5:
462                                         bs := tmp.Bytes()
463                                         var value int
464                                         value, err =
                                            ↪  intCache.Atoi(bs)
465                                         if err == nil && value >
                                            ↪  0 {
466                                                 hab.Steps[l].Disciplines[d].Work
                                                    ↪  =
                                                    ↪  append(hab.Steps[l].Discipli
                                                    ↪  format.DisciplineWorkload{
467                                                     ID:
                                                        ↪  "total_work",
468                                                     Name:
                                                        ↪  "Carga
                                                        ↪  Horária
                                                        ↪  -
                                                        ↪  Total",
469                                                     Unit:
                                                        ↪  "horas",
470                                                     Value:
                                                        ↪  float64(value),
471                                                 })
472                                         }
473                                 case 6:
474                                         var value int
475                                         bs := tmp.Bytes()
476                                         if len(bs) > 0 {
477                                                 value, err =
                                                    ↪  intCache.Atoi(bs)
478                                                 if err == nil &&
                                                    ↪  value > 0 {
```

```
479                                                                 hab.Steps[l].Discip
                                                                    ↪ =
                                                                    ↪ append(hab.Step
                                                                    ↪ format.Discipli
480                                                                     ID:
                                                                        ↪ "total_
481                                                                     Name:
                                                                        ↪ "Carga
                                                                        ↪ Horária
                                                                        ↪ -
                                                                        ↪ Estágio
482                                                                     Unit:
                                                                        ↪ "horas"
483                                                                     Value:
                                                                        ↪ float64
484                                                                 })
485                                                         }
486                                                 }
487                                         case 7:
488                                                 var value int
489                                                 bs := tmp.Bytes()
490                                                 if len(bs) > 0 {
491                                                         value, err =
                                                                ↪ intCache.Atoi(tmp.Bytes
492                                                         if err == nil &&
                                                                ↪ value > 0 {
493                                                                 hab.Steps[l].Discip
                                                                    ↪ =
                                                                    ↪ append(hab.Step
                                                                    ↪ format.Discipli
494                                                                     ID:
                                                                        ↪ "total_
```

```
495                                                                 Name:
                                                                    ↪  "Carga
                                                                    ↪  Horária
                                                                    ↪  -
                                                                    ↪  Práticas
                                                                    ↪  como
                                                                    ↪  Componentes
                                                                    ↪  Curriculares
496                                                                 Unit:
                                                                    ↪  "horas",
497                                                                 Value:
                                                                    ↪  float64(valu
498                                                             })
499                                                     }
500                                             }
501                                     case 8:
502                                             var value int
503                                             bs := tmp.Bytes()
504                                             if len(bs) > 0 {
505                                                     value, err =
                                                        ↪  intCache.Atoi(tmp.Bytes())
506                                                     if err == nil &&
                                                        ↪  value > 0 {
507                                                         hab.Steps[l].Disciplines
                                                            ↪  =
                                                            ↪  append(hab.Steps[l].
                                                            ↪  format.DisciplineWor
508                                                             ID:
                                                                ↪  "total_other
509                                                             Name:
                                                                ↪  "Carga
                                                                ↪  Horária
                                                                ↪  -
                                                                ↪  Atividades
                                                                ↪  Acadêmicos-C
510                                                             Unit:
                                                                ↪  "horas",
511                                                             Value:
                                                                ↪  float64(valu
```

```
512                                                     })
513                                             }
514                                     }
515                             }
516                     case 8:
517                             l := len(hab.Steps) - 1
518                             d :=
                                ↪  len(hab.Steps[l].Disciplines)
                                ↪  - 1
519                             bs :=
                                ↪  bytes.TrimSpace(tmp.Bytes())
520                             if len(bs) == 0 {
521                                     continue
522                             }
523                             switch columnIndex {
524                             case 1:
525                                     index := bytes.Index(bs,
                                        ↪  []byte(" - "))
526                                     var item
                                        ↪  format.DisciplineRelatedItem
527                                     item.Description =
                                        ↪  stringCache.Transform(bs,
                                        ↪  stripSpaces)
528                                     if index > -1 {
529                                             item.Type =
                                                ↪  "discipline_id"
530                                             code :=
                                                ↪  stringCache.Transform(b
                                                ↪  stripSpaces)
531                                             if value, ok :=
                                                ↪  idByCode[code];
                                                ↪  ok {
532                                                     item.Value
                                                        ↪  =
                                                        ↪  value
533                                             } else {
534                                                     item.Type
                                                        ↪  =
                                                        ↪  "discipline_gen
```

```
                        item.Value
                        ↪  =
                        ↪  hash(bs[:index])
                        index =
                        ↪  -1
                }
        }
        if index == -1 {
                item.Type =
                ↪  "text"
                item.Value =
                ↪  item.Description
                item.Description
                ↪  = ""
        }
        var items
        ↪  []format.DisciplineRelatedItem
        dataType :=
        ↪  randomPlaceholder
        if len(bs) != 2 ||
        ↪  !bytes.Equal(bs,
        ↪  []byte("ou")) {
                items =
                ↪  append(items,
                ↪  item)
                dataType = ""
        }
        related :=
        ↪  hab.Steps[l].Disciplines[d].Related
        r := len(related) - 1
        if r >= 0 &&
        ↪  related[r].Type ==
        ↪  randomPlaceholder {
                hab.Steps[l].Disciplines[d].Rela
                ↪  =
                ↪  append(hab.Steps[l].Discipli
                ↪  items...)
        } else {
```

```
555                                 hab.Steps[l].Disciplines[d]
                                    ↪ =
                                    ↪ append(hab.Steps[l].Dis
                                    ↪ format.DisciplineRelate
556                                     Type:
                                        ↪ dataType,
557                                     Items:
                                        ↪ items,
558                                 })
559                         }
560                 case 2:
561                         r :=
                            ↪ len(hab.Steps[l].Disciplines[d]
                            ↪ - 1
562                         hab.Steps[l].Disciplines[d].Related
                            ↪ =
                            ↪ stringCache.Transform(bs,
                            ↪ stripSpaces)
563                         oldDataType :=
                            ↪ hab.Steps[l].Disciplines[d].Rel
564                         if bytes.Contains(bs,
                            ↪ []byte("Requisito"))
                            ↪ {
565                                 hab.Steps[l].Disciplines[d]
                                    ↪ =
                                    ↪ "prerequisite"
566                         } else if
                            ↪ bytes.Contains(bs,
                            ↪ []byte("Conjunto")) {
567                                 hab.Steps[l].Disciplines[d]
                                    ↪ = "set"
568                         } else {
569                                 err =
                                    ↪ newErrUnexpectedRelated
570                         }
```

```
571                                             if err == nil && r > 0 &&
                                                ↪  oldDataType !=
                                                ↪  randomPlaceholder &&
                                                ↪  hab.Steps[l].Disciplines[d].Related[
                                                ↪  ==
                                                ↪  hab.Steps[l].Disciplines[d].Related[
                                                ↪  &&
                                                ↪  hab.Steps[l].Disciplines[d].Related[
                                                ↪  ==
                                                ↪  hab.Steps[l].Disciplines[d].Related[
                                                ↪  {
572                                                     hab.Steps[l].Disciplines[d].Rela
                                                        ↪  =
                                                        ↪  append(hab.Steps[l].Discipli
                                                        ↪  hab.Steps[l].Disciplines[d].
573                                                     hab.Steps[l].Disciplines[d].Rela
                                                        ↪  =
                                                        ↪  hab.Steps[l].Disciplines[d].
574                                             }
575                                     default:
576                                             if err == nil {
577                                                     err =
                                                        ↪  newErrColumn(newErrUnexpecte
                                                        ↪  columnIndex)
578                                             }
579                                     }
580                             case 9:
581                                     if err == nil {
582                                             err =
                                                ↪  newErrExpected(newErrUnexpectedValue
                                                ↪  "")
583                                             if tmp.Len() == 0 {
584                                                     err = nil
585                                             }
586                                     }
587                             }
588                             tmp.Reset()
589                     }
590             } else if tagType == html.TextToken && columnIndex > -1 {
```

```
591                         tmp.Write(bytes.TrimSpace(doc.Text()))
592                 }
593         }
594         if habName != "" && courseName != "" {
595                 hab.Name = habName
596                 hab.Course.Name = courseName
597         }
598         cleanHabilitation(&hab)
599         if err == io.EOF {
600                 err = nil
601         } else {
602                 err = newErrHabilitation(err, &hab)
603         }
604         return hab, err
605 }
```

Arquivo state.go

```
1  package usp2habilitations
2
3  import (
4          "encoding/gob"
5          "errors"
6
7          "github.com/fjorgemota/gurudamatricula/robot/format"
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9  )
10
11 // ErrInvalidStates is returned when invalid states were passed
12 var ErrInvalidStates = errors.New("semantic: Invalid states found (with
   ↪  len=0)")
13
14 type habilitationId struct {
15         CourseID string
16         ID       string
17 }
18
19 func toHabilitationID(hab format.Habilitation) habilitationId {
20         return habilitationId{
21                 CourseID: hab.Course.ID,
```

```
                ID:       hab.ID,
        }
}

// DisciplineState stores state dat aneeded by the discipline crawler
type DisciplineState struct {
        // REQUIRED:
        ID           string
        Habilitation format.Habilitation
        Discipline   *format.Discipline
        // OPTIONAL:
        StringCache pool.StringCache
        IntCache    pool.AtoiCache
}

type HabilitationState struct {
        // REQUIRED:
        ID           string
        Campus       format.Campus
        Habilitation format.Habilitation
        // OPTIONAL:
        StringCache pool.StringCache
        IntCache    pool.AtoiCache
}

type PipelineDisciplineState struct {
        ID                string
        HabilitationIndex int
        StepIndex         int
        DisciplineIndex   int
}

// State stores the data needed by the crawler
type State struct {
        Campus        format.Campus
        Habilitations []format.Habilitation
        Count         int
        Retries       int
        Prefix        string
```

```
61          ID              string
62          Names           []string
63  }
64
65  func init() {
66          gob.Register(State{})
67          gob.Register([]State{})
68  }
```

B.1.18 Pasta robot/usp2offers

Arquivo api.go

```
1   package usp2offers
2
3   import (
4           "fmt"
5           "io"
6           "math/rand"
7           "net/http"
8           "time"
9
10          "github.com/fjorgemota/gurudamatricula/robot/format"
11          robotpool
            ↪  "github.com/fjorgemota/gurudamatricula/robot/format/pool"
12          "github.com/fjorgemota/gurudamatricula/util/clock"
13          "github.com/fjorgemota/gurudamatricula/util/coder"
14          "github.com/fjorgemota/gurudamatricula/util/pool"
15          "github.com/sirupsen/logrus"
16          "golang.org/x/net/html"
17  )
18
19  func randomSleep(clk clock.Clock, delay time.Duration) {
20          clk.Sleep(time.Duration((rand.Float64() + 0.5) * float64(delay)))
21  }
22
23  func FetchCampi(logger logrus.FieldLogger, client *http.Client, clk
    ↪  clock.Clock, options Options, campi []format.Campus, ID string)
    ↪  ([]State, error) {
24          var result []State
```

```
25          var err error
26          for _, campus := range campi {
27                  var disciplines []format.DisciplineOffer
28                  if err == nil {
29                          randomSleep(clk, options.DelayBetweenRequests)
30                          disciplines, err = getDisciplines(client, campus,
                            ↪  ID, options.UserAgent)
31                  }
32                  logger.WithField("num_disciplines",
                    ↪  len(disciplines)).WithField("campus",
                    ↪  campus.Name).Debug("Recognizing disciplines")
33                  if err == nil {
34                          result = append(result, State{
35                                  Disciplines: disciplines,
36                                  Campus:      campus,
37                                  ID:          ID,
38                                  Prefix:      fmt.Sprintf("%s-%s-%d", ID,
                                    ↪  campus.ID, clk.Now().Unix()),
39                          })
40                  }
41                  clk.Sleep(options.DelayBetweenRequests)
42          }
43          return result, err
44  }
45
46  func BalanceDisciplines(logger logrus.FieldLogger, clk clock.Clock,
    ↪  parallelRequests int, dispatch func([]State) error, states
    ↪  ...[]State) error {
47          var allStates []State
48          for _, state := range states {
49                  allStates = append(allStates, state...)
50          }
51          disciplineCount := 0
52          for _, state := range allStates {
53                  disciplineCount += len(state.Disciplines)
54          }
55          var batch []State
56          var actualState State
57          splitPoint := disciplineCount / parallelRequests
```

```
58          logger.WithField("disciplines_count",
            ↪  disciplineCount).WithField("splitPoint",
            ↪  splitPoint).Debug("Detecting disciplines to balance")
59          count := 0
60          var err error
61          for _, selectedState := range allStates {
62                  if selectedState.Campus != actualState.Campus {
63                          if len(actualState.Disciplines) > 0 {
64                                  batch = append(batch, actualState)
65                          }
66                          actualState = State{}
67                          actualState.Campus = selectedState.Campus
68                          actualState.ID = selectedState.ID
69                          actualState.Prefix = selectedState.Prefix
70                  }
71                  for _, discipline := range selectedState.Disciplines {
72                          if count > splitPoint && err == nil {
73                                  batch = append(batch, actualState)
74                                  err = dispatch(batch)
75                                  batch = batch[:0]
76                                  actualState = State{}
77                                  actualState.Campus = selectedState.Campus
78                                  actualState.ID = selectedState.ID
79                                  actualState.Prefix = selectedState.Prefix
80                                  count = 0
81                          }
82                          actualState.Disciplines =
                            ↪  append(actualState.Disciplines, discipline)
83                          count++
84                  }
85          }
86          if count > 0 && err == nil {
87                  batch = append(batch, actualState)
88                  err = dispatch(batch)
89          }
90          return err
91  }
92
```

```
93  func FetchDiscipline(client *http.Client, encoder coder.StreamingEncoder,
    ↪  clk clock.Clock, options Options, actualState DisciplineState) error
    ↪  {
94          var err error
95          stringCache := actualState.StringCache
96          if stringCache == nil {
97                  stringCache = pool.GetStringCache()
98                  defer pool.PutStringCache(stringCache)
99          }
100         intCache := actualState.IntCache
101         if intCache == nil {
102                 intCache = pool.GetAtoiCache()
103                 defer pool.PutAtoiCache(intCache)
104         }
105         var reader io.Reader
106         if err == nil {
107                 randomSleep(clk, options.DelayBetweenRequests)
108                 reader, err = fetchDiscipline(client,
                    ↪  actualState.Discipline, actualState.ID,
                    ↪  options.UserAgent)
109         }
110         if err == nil {
111                 doc := html.NewTokenizer(reader)
112                 actualState.Teams, err = parse(actualState.Teams[:0], doc,
                    ↪  intCache, stringCache, actualState.Discipline,
                    ↪  actualState.Campus)
113         }
114         for _, team := range actualState.Teams {
115                 if err == nil {
116                         err = encoder.Encode(team)
117                 }
118                 if err == nil {
119                         robotpool.PutTeam(team)
120                 }
121         }
122         return err
123 }
```

Arquivo bot.go

```
1  package usp2offers
2
3  import (
4          "crypto/sha1"
5          "fmt"
6          "io"
7          "net/http"
8
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10         "github.com/fjorgemota/gurudamatricula/robot/joiner"
11         "github.com/fjorgemota/gurudamatricula/robot/util/uspreq"
12         "github.com/fjorgemota/gurudamatricula/util/clock"
13         "github.com/fjorgemota/gurudamatricula/util/file"
14         "github.com/fjorgemota/gurudamatricula/util/pipeline"
15         "github.com/fjorgemota/gurudamatricula/util/pool"
16         "github.com/sirupsen/logrus"
17         "golang.org/x/net/context"
18 )
19
20 // Handler defines a few constructors that Bot needs to work
21 type Handler interface {
22         joiner.Handler
23         GetHTTPClient(ctx context.Context, jar http.CookieJar)
           ↪  *http.Client
24         GetManager(ctx context.Context) (file.Manager, error)
25         GetLogger(ctx context.Context) logrus.FieldLogger
26 }
27
28 type bot struct {
29         options Options
30         pipe    pipeline.Pipeline
31         handler Handler
32         clk     clock.Clock
33 }
34
35 func (b bot) start(ctx context.Context, pipe pipeline.Pipeline, target
   ↪  string) (pipeline.FutureID, error) {
36         client := b.handler.GetHTTPClient(ctx, nil)
37         logger := b.handler.GetLogger(ctx)
```

```
38          var results []pipeline.FutureID
39          err := uspreq.Start(logger, client, func(campi []format.Campus,
            ↪  ID string) error {
40                  futIDs, errDispatch :=
                    ↪  pipe.DispatchWithOptions(b.options.Tasks.FetchCampi,
                    ↪  pipeline.TaskOptions{
41                          MaxAttempts: 1,
42                  }, campi, ID)
43                  if errDispatch == nil {
44                          results = append(results, futIDs[0])
45                  }
46                  return errDispatch
47          }, b.options.ParallelRequests, b.options.UserAgent)
48          if err == nil {
49                  args := make([]interface{}, 0, len(results)+1)
50                  args = append(args, target)
51                  for _, result := range results {
52                          args = append(args, result)
53                  }
54                  results, err = pipe.Dispatch(b.options.Tasks.BalanceCampi,
                    ↪  args...)
55          }
56          var result pipeline.FutureID
57          if err == nil {
58                  result = results[0]
59          }
60          return result, err
61  }
62
63  func (b bot) Start(pipe pipeline.Pipeline, target string)
    ↪  (pipeline.FutureID, error) {
64          futIDs, err := pipe.DispatchWithOptions(b.options.Tasks.Start,
            ↪  pipeline.TaskOptions{
65                  MaxAttempts: 1,
66          }, target)
67          var futID pipeline.FutureID
68          if len(futIDs) > 0 {
69                  futID = futIDs[0]
70          }
```

```
71          return futID, err
72  }
73
74  func (b bot) balanceDisciplines(ctx context.Context, pipe
    ↪  pipeline.Pipeline, target string, states ...[]State)
    ↪  (pipeline.FutureID, error) {
75          logger := b.handler.GetLogger(ctx)
76          var results []pipeline.FutureID
77          err := BalanceDisciplines(logger, b.clk,
            ↪  b.options.ParallelRequests, func(batch []State) error {
78                  result, err := pipe.Dispatch(b.options.Tasks.FetchData,
                    ↪  batch)
79                  if err == nil {
80                          results = append(results, result[0])
81                  }
82                  return err
83          }, states...)
84          var result pipeline.FutureID
85          if err == nil {
86                  files := make([]interface{}, 0, len(results)+1)
87                  files = append(files, target)
88                  for _, result := range results {
89                          files = append(files, result)
90                  }
91                  results, err = pipe.Dispatch(b.options.Tasks.JoinFiles,
                    ↪  files...)
92          }
93          if err == nil {
94                  result = results[0]
95          }
96          return result, err
97  }
98
99  func (b bot) generateName(actualState State) string {
100         var target []byte
101         for index := 0; index < b.options.BatchSize && index <
            ↪  len(actualState.Disciplines); index++ {
102                 target = append(target,
                    ↪  actualState.Disciplines[index].ID...)
```

```
103         }
104         now := b.clk.Now()
105         return fmt.Sprintf("%s-%x-%d-%d.partial_1", actualState.Prefix,
            ↪  sha1.Sum(target), now.Unix(), now.Nanosecond())
106 }
107
108 func (b bot) joinFiles(ctx context.Context, pipe pipeline.Pipeline,
    ↪  target string, filenames ...string) (string, error) {
109         var err error
110         result := filenames[0]
111         if len(filenames) > 1 {
112                 // We just return the file we have, so we can avoid
                    ↪  writing a new file
113                 var manager file.Manager
114                 manager, err = b.handler.GetManager(ctx)
115                 if err == nil {
116                         err = joiner.JoinTeams(manager, b.handler,
                            ↪  filenames, target)
117                 }
118                 result = target
119         }
120         return result, err
121 }
122 func (b bot) fetchCampi(ctx context.Context, pipe pipeline.Pipeline,
    ↪  campi []format.Campus, ID string) ([]State, error) {
123         logger := b.handler.GetLogger(ctx)
124         client := b.handler.GetHTTPClient(ctx, nil)
125         return FetchCampi(logger, client, b.clk, b.options, campi, ID)
126 }
127
128 func (b bot) fetchData(ctx context.Context, pipe pipeline.Pipeline,
    ↪  states []State) (pipeline.FutureID, error) {
129         logger := b.handler.GetLogger(ctx)
130         client := b.handler.GetHTTPClient(ctx, nil)
131         err := ErrInvalidStates
132         if len(states) > 0 {
133                 err = nil
134         }
135         disciplineCount := 0
```

```
136         for _, state := range states {
137                 disciplineCount += len(state.Disciplines)
138         }
139         stateIndex := 0
140         for ; stateIndex < len(states); stateIndex++ {
141                 if len(states[stateIndex].Disciplines) > 0 {
142                         break
143                 }
144         }
145         logger.WithField("discipline_count",
            ↪  disciplineCount).WithField("campus",
            ↪  states[stateIndex].Campus.Name).Debug("Processing
            ↪  disciplines")
146         name := b.generateName(states[stateIndex])
147         var manager file.Manager
148         if err == nil {
149                 manager, err = b.handler.GetManager(ctx)
150         }
151         var writer io.WriteCloser
152         if err == nil {
153                 writer, err = manager.Writer(name)
154         }
155         intCache := pool.GetAtoiCache()
156         defer pool.PutAtoiCache(intCache)
157         stringCache := pool.GetStringCache()
158         defer pool.PutStringCache(stringCache)
159         var index int
160         if err == nil {
161                 encoder := b.handler.GetStreamingEncoder(writer)
162                 actualState := states[stateIndex]
163                 teams := make([]*format.Team, 0, 100)
164                 for c := 0; err == nil && c < b.options.BatchSize &&
                    ↪  index < len(actualState.Disciplines); c++ {
165                         err = FetchDiscipline(client, encoder, b.clk,
                            ↪  b.options, DisciplineState{
166                                 Campus:     actualState.Campus,
167                                 Discipline:
                                    ↪  actualState.Disciplines[index],
168                                 ID:         actualState.ID,
```

```
                                IntCache:    intCache,
                                StringCache: stringCache,
                                Teams:       teams,
                        })
                        if err == nil {
                                index++
                        }
                }
                if err == nil {
                        err = encoder.Flush()
                }
                if err == nil {
                        err = encoder.Close()
                }
        }
        shouldDispatch := false
        if err == nil {
                states[stateIndex].Names =
                ↪  append(states[stateIndex].Names, name)
        } else {
                states[stateIndex].Retries++
                shouldDispatch = states[stateIndex].Retries <
                ↪  b.options.MaxRetries
                if shouldDispatch {
                        logger.WithError(err).Debug("Retrying again after
                        ↪  error detected")
                        // Discard error because we will retry manually
                        err = nil
                }
                if err == nil {
                        err = manager.Delete(name)
                }
        }
        for stateIndex < len(states) && !shouldDispatch && err == nil {
                shouldDispatch = index <
                ↪  len(states[stateIndex].Disciplines)
                states[stateIndex].Disciplines =
                ↪  states[stateIndex].Disciplines[index:]
                if !shouldDispatch {
```

```
203                         stateIndex++
204                         index = 0
205                 }
206         }
207         var result pipeline.FutureID
208         var results []pipeline.FutureID
209         if err == nil {
210                 if shouldDispatch {
211                         results, err =
                        ↪  pipe.Dispatch(b.options.Tasks.FetchData,
                        ↪  states)
212                 } else {
213                         var names []string
214                         hasher := pool.GetSha1()
215                         for _, stateObj := range states {
216                                 names = append(names, stateObj.Names...)
217                                 if err == nil {
218                                         _, err = io.WriteString(hasher,
                                        ↪  stateObj.Prefix)
219                                 }
220                         }
221                         files := make([]interface{}, 0, len(names)+1)
222                         now := b.clk.Now()
223                         files = append(files,
                        ↪  fmt.Sprintf("%x-%d-%d.partial_2",
                        ↪  hasher.Sum(nil), now.Unix(),
                        ↪  now.Nanosecond()))
224                         pool.PutSha1(hasher)
225                         for _, name := range names {
226                                 files = append(files, name)
227                         }
228                         if err == nil {
229                                 results, err =
                                ↪  pipe.Dispatch(b.options.Tasks.JoinFiles,
                                ↪  files...)
230                         }
231                 }
232         }
233         if len(results) > 0 {
```

```
234                 result = results[0]
235         }
236         return result, err
237 }
238
239 // Bot defines the basic interface to start the processing of the data
240 type Bot interface {
241         Start(pipe pipeline.Pipeline, target string) (pipeline.FutureID,
            ↪  error)
242 }
243
244 // NewBot Gets a new Bot
245 func NewBot(disp pipeline.Dispatcher, clk clock.Clock, handler Handler,
    ↪  opt Options) (Bot, error) {
246         opt = getDefaults(opt)
247         instance := &bot{
248                 options: opt,
249                 handler: handler,
250                 clk:     clk,
251         }
252         err := disp.Register(opt.Tasks.FetchData, instance.fetchData)
253         if err == nil {
254                 err = disp.Register(opt.Tasks.Start, instance.start)
255         }
256         if err == nil {
257                 err = disp.Register(opt.Tasks.FetchCampi,
                    ↪  instance.fetchCampi)
258         }
259         if err == nil {
260                 err = disp.Register(opt.Tasks.BalanceCampi,
                    ↪  instance.balanceDisciplines)
261         }
262         if err == nil {
263                 err = disp.Register(opt.Tasks.JoinFiles,
                    ↪  instance.joinFiles)
264         }
265         return instance, err
266 }
```

Arquivo fetcher.go

```
1  package usp2offers
2
3  import (
4          "bytes"
5          "io"
6          "net/http"
7          "net/url"
8
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10         "github.com/fjorgemota/gurudamatricula/robot/util/uspreq"
11         "github.com/fjorgemota/gurudamatricula/util/pool"
12         "golang.org/x/net/html"
13         "golang.org/x/net/html/atom"
14 )
15
16 const disciplinesIndex =
   ↪  "https://uspdigital.usp.br/jupiterweb/jupDisciplinaLista"
17
18 const teamsIndex = "https://uspdigital.usp.br/jupiterweb/obterTurma"
19
20 func getDisciplines(client *http.Client, campus format.Campus, ID,
   ↪  userAgent string) ([]format.DisciplineOffer, error) {
21         var disciplines []format.DisciplineOffer
22         req, err := uspreq.GetRequest(disciplinesIndex, url.Values{
23                 "tipo":  []string{"D"},
24                 "codcg": []string{campus.ID},
25                 "letra": []string{"A-Z"},
26         }, userAgent)
27         var resp *http.Response
28         if err == nil {
29                 resp, err = client.Do(req)
30         }
31         var reader io.Reader
32         if err == nil {
33                 reader, err = uspreq.GetDoc(req, resp, ID)
34         }
35         if err == nil {
```

```
36                  doc := html.NewTokenizer(reader)
37                  columnIndex := -1
38                  columnCount := -1
39                  tmp := pool.GetBytesBuffer()
40                  defer pool.PutBytesBuffer(tmp)
41                  var discipline format.DisciplineOffer
42                  for err == nil {
43                          tagType := doc.Next()
44                          if tagType == html.ErrorToken {
45                                  err = doc.Err()
46                          } else if tagType == html.StartTagToken {
47                                  tag, _ := doc.TagName()
48                                  tagName := atom.String(tag)
49                                  switch tagName {
50                                  case "table":
51                                          columnIndex = 0
52                                  case "tr":
53                                          columnIndex = 0
54                                  case "td":
55                                          columnIndex++
56                                  }
57                          } else if tagType == html.EndTagToken {
58                                  tag, _ := doc.TagName()
59                                  tagName := atom.String(tag)
60                                  switch tagName {
61                                  case "table":
62                                          columnIndex = -1
63                                          columnCount = -1
64                                  case "tr":
65                                          if columnCount == 4 {
66                                                  if discipline.Code != ""
                                                    ↪  && discipline.Name !=
                                                    ↪  "" {
67                                                          disciplines =
                                                            ↪  append(disciplines,
                                                            ↪  discipline)
68                                                  }
69                                                  discipline =
                                                    ↪  format.DisciplineOffer{}
```

```
70                                      }
71                                      columnCount = columnIndex
72                                  case "td":
73                                      if columnCount != 4 ||
                                        ↪  columnIndex >= 5 || tmp.Len()
                                        ↪  == 0 {
74                                          continue
75                                      }
76                                      switch columnIndex {
77                                      case 1:
78                                          discipline.ID =
                                            ↪  hash(tmp.Bytes())
79                                          discipline.GenericID =
                                            ↪  discipline.ID
80                                          discipline.Code =
                                            ↪  tmp.String()
81                                      case 2:
82                                          discipline.Name =
                                            ↪  tmp.String()
83                                      }
84                                      tmp.Reset()
85                                  }
86                              } else if tagType == html.TextToken &&
                                ↪  columnCount == 4 {
87                                  tmp.Write(bytes.TrimSpace(doc.Text()))
88                              }
89                      }
90                      if err == io.EOF {
91                              err = nil
92                      }
93              }
94              return disciplines, err
95      }
96
97      func fetchDiscipline(client *http.Client, discipline
        ↪  format.DisciplineOffer, ID, userAgent string) (io.Reader, error) {
98              req, err := uspreq.GetRequest(teamsIndex, url.Values{
99                      "sgldis": []string{discipline.Code},
100             }, userAgent)
```

```
101         var resp *http.Response
102         if err == nil {
103                 resp, err = client.Do(req)
104         }
105         var reader io.Reader
106         if err == nil {
107                 reader, err = uspreq.GetDoc(req, resp, ID)
108         }
109         return reader, err
110 }
```

Arquivo options.go

```
1  package usp2offers
2
3  import (
4          "runtime"
5          "time"
6  )
7
8  type Tasks struct {
9          Start        string
10         FetchCampi   string
11         BalanceCampi string
12         FetchData    string
13         JoinFiles    string
14 }
15
16 // Options define some Options that should be defined in compile-time
17 type Options struct {
18         UserAgent            string
19         DelayBetweenRequests time.Duration
20         BatchSize            int
21         ParallelRequests     int
22         MaxRetries           int
23         Tasks                Tasks
24 }
25
26 func getDefaults(opt Options) Options {
27         if opt.Tasks.FetchData == "" {
```

```
28                  opt.Tasks.FetchData = "usp.fetch_data"
29          }
30          if opt.Tasks.FetchCampi == "" {
31                  opt.Tasks.FetchCampi = "usp.fetch_disciplines"
32          }
33          if opt.Tasks.JoinFiles == "" {
34                  opt.Tasks.JoinFiles = "usp.join"
35          }
36          if opt.Tasks.BalanceCampi == "" {
37                  opt.Tasks.BalanceCampi = "usp.balance"
38          }
39          if opt.Tasks.Start == "" {
40                  opt.Tasks.Start = "usp.start"
41          }
42          if opt.UserAgent == "" {
43                  opt.UserAgent = "Mozilla/5.0 (X11; Linux x86_64)
                    ↪  AppleWebKit/537.36 (KHTML, like Gecko)
                    ↪  Chrome/60.0.3112.101 Safari/537.36"
44          }
45          if opt.BatchSize <= 0 {
46                  opt.BatchSize = 10
47          }
48          if opt.DelayBetweenRequests <= 0 {
49                  opt.DelayBetweenRequests = 2 * time.Second
50          }
51          if opt.ParallelRequests <= 0 {
52                  opt.ParallelRequests = runtime.NumCPU()
53          }
54          if opt.MaxRetries == 0 {
55                  opt.MaxRetries = 3
56          }
57          if opt.MaxRetries < 0 {
58                  opt.MaxRetries = 0
59          }
60          return opt
61  }
```

Arquivo parser.go

```
1  package usp2offers
```

```
2
3  import (
4          "bytes"
5          "crypto/sha1"
6          "encoding/hex"
7          "fmt"
8          "io"
9          "sort"
10         "strings"
11
12         "github.com/davecgh/go-spew/spew"
13
14         "github.com/fjorgemota/gurudamatricula/robot/format"
15         "github.com/fjorgemota/gurudamatricula/robot/util/attrs"
16         robotpool
           ↪  "github.com/fjorgemota/gurudamatricula/robot/format/pool"
17         "github.com/fjorgemota/gurudamatricula/util/pool"
18         "github.com/pkg/errors"
19         "golang.org/x/net/html"
20         "golang.org/x/net/html/atom"
21 )
22
23 func cleanTables(actualTeam *format.Team) {
24         tables := make([]format.Table, 0, len(actualTeam.Tables))
25         for _, table := range actualTeam.Tables {
26                 if len(table.Rows) > 0 {
27                         rows := table.Rows
28                         for i, row := range rows {
29                                 for key, value := range row {
30                                         if len(value) == 0 {
31                                                 delete(row, key)
32                                         }
33                                 }
34                                 rows[i] = row
35                         }
36                         table.Rows = rows
37                         tables = append(tables, table)
38                 }
39         }
```

```
40          actualTeam.Tables = tables
41  }
42
43  func setTables(actualTeam *format.Team) {
44          var schedules format.Table
45          schedules.ID = "schedules"
46          schedules.Title = "Horários"
47          schedules.Columns = []format.Column{{ID: "schedule", Name:
            ↪  "Horários"}, {ID: "teacher", Name: "Professor"}}
48          var vacancies format.Table
49          vacancies.ID = "vacancies"
50          vacancies.Title = "Vagas"
51          vacancies.Columns = []format.Column{{ID: "type", Name: "Tipo"},
            ↪  {ID: "course", Name: "Curso"}, {ID: "vacancies", Name:
            ↪  "Vagas"}, {ID: "enrolled", Name: "Matriculados"}, {ID:
            ↪  "pending", Name: "Pendentes"}, {ID: "subscribed", Name:
            ↪  "Inscritos"}}
52          var activities format.Table
53          activities.ID = "activities"
54          activities.Title = "Atividades"
55          activities.Columns = []format.Column{{ID: "teachers", Name:
            ↪  "Professores"}, {ID: "type", Name: "Tipo de Atividade"}, {ID:
            ↪  "total", Name: "Carga Horária (total)"}}
56          var details format.Table
57          details.ID = "details"
58          details.Title = "Detalhes"
59          details.Columns = []format.Column{{ID: "key", Name: ""}, {ID:
            ↪  "value", Name: ""}}
60          actualTeam.Tables = append(actualTeam.Tables, schedules)
61          actualTeam.Tables = append(actualTeam.Tables, vacancies)
62          actualTeam.Tables = append(actualTeam.Tables, activities)
63          actualTeam.Tables = append(actualTeam.Tables, details)
64  }
65
66  func hash(bs []byte) string {
67          result := sha1.Sum(bs)
68          return hex.EncodeToString(result[:])
69  }
70
```

```
71 func parse(teams []*format.Team, doc *html.Tokenizer, intCache
   ↪ pool.AtoiCache, stringCache pool.StringCache, discipline
   ↪ format.DisciplineOffer, campus format.Campus) ([]*format.Team, error)
   ↪ {
72         type vacancyData struct {
73                 Type            string
74                 Course          string
75                 VacanciesNumber int
76                 EnrolledNumber  int
77                 Vacancies       string
78                 Subscribed      string
79                 Pending         string
80                 Enrolled        string
81         }
82         type activityData struct {
83                 Teachers []string
84                 Type     string
85                 Workload string
86         }
87         var activity activityData
88         columnIndex := -1
89         infoType := -1
90         rowIndex := -1
91         tableType := -1
92         var err error
93         tmp := pool.GetBytesBuffer()
94         attrTmp := pool.GetBytesBuffer()
95         defer pool.PutBytesBuffer(tmp)
96         defer pool.PutBytesBuffer(attrTmp)
97         team := robotpool.GetTeam()
98         setTables(team)
99         var schedule format.Schedule
100        var vacancy vacancyData
101        teachers := make(map[format.Teacher]struct{}, 0)
102        for err == nil {
103                tagType := doc.Next()
104                if tagType == html.ErrorToken {
105                        err = doc.Err()
106                } else if tagType == html.StartTagToken {
```

```
107                         tag, hasNextAttr := doc.TagName()
108                         tagName := atom.String(tag)
109                         switch tagName {
110                         case "table":
111                                 columnIndex = 0
112                                 rowIndex = 0
113                         case "tr":
114                                 columnIndex = 0
115                                 rowIndex++
116                         case "td":
117                                 if infoType == 11 {
118                                         var attr []byte
119                                         attr, err =
                                            ↪  attrs.GetAttrBytes(doc,
                                            ↪  "colspan", hasNextAttr,
                                            ↪  attrTmp)
120                                         if bytes.Equal(attr, []byte("2"))
                                            ↪  {
121                                                 // If we have colspan=2,
                                                    ↪  we need to set
                                                    ↪  Disciplines
122                                                 // to an empty record too
123                                                 infoType = 10
124                                         }
125                                 }
126                                 columnIndex++
127                         }
128                 } else if tagType == html.EndTagToken {
129                         tag, _ := doc.TagName()
130                         tagName := atom.String(tag)
131                         switch tagName {
132                         case "table":
133                                 if infoType == 11 && tableType == 3 {
134                                         // We are in the vacancies table
                                            ↪  and finishing it...
135                                         // It's fair to just finish the
                                            ↪  Team here =)
136                                         team.Campus = campus
137                                         team.Discipline = discipline
```

```
                                for teacher := range teachers {
                                        if len(teacher.Name) > 0 {
                                                team.Teachers = append(team.Teachers, teacher)
                                        }
                                        delete(teachers, teacher)
                                }
                                sort.Slice(team.Teachers, func(i int, j int) bool {
                                        return team.Teachers[i].Name < team.Teachers[j].Name
                                })
                                team.Course.Exclusive = format.Unknown
                                cleanTables(team)
                                teams = append(teams, team)
                                team = robotpool.GetTeam()
                                setTables(team)
                        }
                        columnIndex = -1
                        tableType = -1
                        infoType = -1
                        rowIndex = -1
                case "tr":
                        if infoType == 11 && vacancy.Type != "" {
                                team.Vacancies.Offered += vacancy.VacanciesNumber
                                team.Vacancies.Filled += vacancy.EnrolledNumber
```

```
161                                     team.Tables[1].Rows =
                                        ↪  append(team.Tables[1].Rows,
                                        ↪  map[string]string{"type":
                                        ↪  vacancy.Type, "course":
                                        ↪  vacancy.Course, "vacancies":
                                        ↪  vacancy.Vacancies, "enrolled":
                                        ↪  vacancy.Enrolled, "pending":
                                        ↪  vacancy.Pending, "subscribed":
                                        ↪  vacancy.Subscribed})
162                                     typ := vacancy.Type
163                                     vacancy = vacancyData{}
164                                     vacancy.Type = typ
165                                 } else if infoType == 6 &&
                                    ↪  schedule.DayOfWeek != 0 {
166                                     team.Schedules =
                                        ↪  append(team.Schedules,
                                        ↪  schedule)
167                                     schedule = format.Schedule{}
168                                 } else if infoType == 17 && tableType ==
                                    ↪  4 && activity.Type != "" {
169                                     team.Tables[2].Rows =
                                        ↪  append(team.Tables[2].Rows,
                                        ↪  map[string]string{"teachers":
                                        ↪  strings.Join(activity.Teachers,
                                        ↪  ", "), "type": activity.Type,
                                        ↪  "total": activity.Workload})
170                                     activity = activityData{}
171                                 }
172                             case "td":
173                                 bs := tmp.Bytes()
174                                 switch infoType {
175                                 case -1:
176                                     if columnIndex == 1 {
177                                         if tableType == -1 {
178                                             if
                                                ↪  bytes.Contains(bs,
                                                ↪  []byte("Horário"))
                                                ↪  {
```

```
179                                         tableType
                                            ↪  = 2
180                                         infoType
                                            ↪  = 5
181                                     } else if
                                        ↪  bytes.Contains(bs,
                                        ↪  []byte("Código"))
                                        ↪  &&
182                                         bytes.Contains(bs,
                                            ↪  []byte("Turma"))
                                            ↪  {
183                                         infoType
                                            ↪  = 1
184                                         tableType
                                            ↪  = 1
185                                     } else if
                                        ↪  bytes.Contains(bs,
                                        ↪  []byte("Atividades"))
                                        ↪  &&
186                                         bytes.Contains(bs,
                                            ↪  []byte("Didáticas"))
                                            ↪  {
187                                         infoType
                                            ↪  = 17
188                                         tableType
                                            ↪  = 4
189                                     }
190                                 } else if tableType == 1
                                    ↪  {
191                                         if
                                            ↪  bytes.Contains(bs,
                                            ↪  []byte("Início"))
                                            ↪  {
192                                         infoType
                                            ↪  = 2
193                                     } else if
                                        ↪  bytes.Contains(bs,
                                        ↪  []byte("Fim"))
                                        ↪  {
```

```
194                                                     infoType
                                                        ↪  = 3
195                                             } else if
                                                ↪  bytes.Contains(bs,
                                                ↪  []byte("Tipo"))
                                                ↪  {
196                                                     infoType
                                                        ↪  = 4
197                                             } else if
                                                ↪  bytes.Contains(bs,
                                                ↪  []byte("Observações"))
                                                ↪  {
198                                                     infoType
                                                        ↪  = 20
199                                             }
200                                     }
201                                 } else if columnIndex == 2 {
202                                     if tableType == -1 {
203                                         if
                                            ↪  bytes.Contains(bs,
                                            ↪  []byte("Vagas"))
                                            ↪  {
204                                             tableType
                                                ↪  = 3
205                                             infoType
                                                ↪  = 11
206                                         }
207                                     }
208                                 }
209                             case 1:
210                                 if tableType == 1 {
211                                     index :=
                                        ↪  bytes.IndexByte(bs, '
                                        ↪  ')
212                                     if index > 0 {
213                                         bs = bs[:index]
214                                     }
215                                     team.Code =
                                        ↪  stringCache.String(bs)
```

```
                        team.ID = hash(bs)
                        period := team.Code[:5]
                        team.Period.ID =
                        ↪ hash(bs[:5])
                        team.Period.Name =
                        ↪ period[:4] + "-" +
                        ↪ string(period[4])
                        infoType = -1
                }
        case 2:
                if tableType == 1 {
                        team.Tables[3].Rows =
                        ↪ append(team.Tables[3].Rows,
                        ↪ map[string]string{"key":
                        ↪ "Data de Inicio",
                        ↪ "value":
                        ↪ stringCache.String(bs)})
                        infoType = -1
                }
        case 3:
                if tableType == 1 {
                        team.Tables[3].Rows =
                        ↪ append(team.Tables[3].Rows,
                        ↪ map[string]string{"key":
                        ↪ "Data Final", "value":
                        ↪ stringCache.String(bs)})
                        infoType = -1
                }
        case 4:
                if tableType == 1 {
                        team.Tables[3].Rows =
                        ↪ append(team.Tables[3].Rows,
                        ↪ map[string]string{"key":
                        ↪ "Tipo", "value":
                        ↪ stringCache.String(bs)})
                        infoType = -1
                }
        case 5:
```

```
238                                 if tableType == 2 &&
                                    ↪  bytes.Contains(bs,
                                    ↪  []byte("Prof")) {
239                                         infoType = 6
240                                 }
241                         case 6:
242                                 if tableType == 2 {
243                                         if len(bs) > 0 {
244                                                 switch
                                                    ↪  stringCache.String(bs)
                                                    ↪  {
245                                                 case "dom":
246                                                         schedule.DayOfWeek
                                                            ↪  = 1
247                                                 case "seg":
248                                                         schedule.DayOfWeek
                                                            ↪  = 2
249                                                 case "ter":
250                                                         schedule.DayOfWeek
                                                            ↪  = 3
251                                                 case "qua":
252                                                         schedule.DayOfWeek
                                                            ↪  = 4
253                                                 case "qui":
254                                                         schedule.DayOfWeek
                                                            ↪  = 5
255                                                 case "sex":
256                                                         schedule.DayOfWeek
                                                            ↪  = 6
257                                                 case "sab":
258                                                         schedule.DayOfWeek
                                                            ↪  = 7
259                                                 }
260                                         } else if
                                            ↪  len(team.Schedules) >
                                            ↪  0 {
```

```
261                                             // This will
                                                ↪  panic if
                                                ↪  USP's site
                                                ↪  emits an
                                                ↪  empty day of
                                                ↪  week without
262                                             // a schedule
                                                ↪  before, but
                                                ↪  it's a
                                                ↪  problem for
                                                ↪  us anyway
263                                             schedule.DayOfWeek
                                                ↪  =
                                                ↪  team.Schedules[len(team.Sche
264                                         }
265                                         if schedule.DayOfWeek ==
                                            ↪  0 && err == nil {
266                                             err =
                                                ↪  errors.Errorf("Error:
                                                ↪  Expected Day
                                                ↪  Of Week,
                                                ↪  found '%s'",
                                                ↪  stringCache.String(bs))
267                                         }
268                                         infoType = 7
269                                 }
270                             case 7:
271                                 if tableType == 2 {
272                                     if len(bs) > 0 {
273                                         var hour, minute
                                            ↪  int
274                                         bs =
                                            ↪  bytes.TrimSpace(bs)
275                                         separator :=
                                            ↪  bytes.IndexByte(bs,
                                            ↪  ':')
276                                         hour, err =
                                            ↪  intCache.Atoi(bs[:separator]
277                                         if err == nil {
```

```
278                                                     minute,
                                                        ↪  err =
                                                        ↪  intCache.Atoi(b
279                                                 }
280                                                 if err == nil {
281                                                     schedule.Start.Hour
                                                        ↪  =
                                                        ↪  int8(hour)
282                                                     schedule.Start.Minu
                                                        ↪  =
                                                        ↪  int8(minute)
283                                                 }
284                                             }
285                                             infoType = 8
286                                         }
287                                 case 8:
288                                         if tableType == 2 {
289                                             if len(bs) > 0 {
290                                                 var hour, minute
                                                    ↪  int
291                                                 bs =
                                                    ↪  bytes.TrimSpace(bs)
292                                                 separator :=
                                                    ↪  bytes.IndexByte(bs,
                                                    ↪  ':')
293                                                 hour, err =
                                                    ↪  intCache.Atoi(bs[:separ
294                                                 if err == nil {
295                                                     minute,
                                                        ↪  err =
                                                        ↪  intCache.Atoi(b
296                                                 }
297                                                 if err == nil {
298                                                     schedule.End.Hour
                                                        ↪  =
                                                        ↪  int8(hour)
299                                                     schedule.End.Minute
                                                        ↪  =
                                                        ↪  int8(minute)
```

```
                                        }
                                }
                                infoType = 9
                        }
                case 9:
                        if tableType == 2 {
                                teacher :=
                                ↪  format.Teacher{
                                        Name:
                                        ↪  stringCache.String(bs),
                                }
                                teacher.ID = hash(bs)
                                teachers[teacher] =
                                ↪  struct{}{}
                                var dayOfWeek string
                                switch schedule.DayOfWeek
                                ↪  {
                                case 1:
                                        dayOfWeek =
                                        ↪  "domingo"
                                case 2:
                                        dayOfWeek =
                                        ↪  "segunda-feira"
                                case 3:
                                        dayOfWeek =
                                        ↪  "terça-feira"
                                case 4:
                                        dayOfWeek =
                                        ↪  "quarta-feira"
                                case 5:
                                        dayOfWeek =
                                        ↪  "quinta-feira"
                                case 6:
                                        dayOfWeek =
                                        ↪  "sexta-feira"
                                case 7:
                                        dayOfWeek =
                                        ↪  "sábado"
                                }
```

```
328                                     scheduleText :=
                                        ↪ fmt.Sprintf("%s,
                                        ↪ %02d:%02d até
                                        ↪ %02d:%02d", dayOfWeek,
                                        ↪ schedule.Start.Hour,
                                        ↪ schedule.Start.Minute,
                                        ↪ schedule.End.Hour,
                                        ↪ schedule.End.Minute)
329                                     team.Tables[0].Rows =
                                        ↪ append(team.Tables[0].Rows,
                                        ↪ map[string]string{"schedule":
                                        ↪ scheduleText,
                                        ↪ "teacher":
                                        ↪ teacher.Name})
330                                     infoType = 6
331                             }
332                     case 10:
333                             if rowIndex != 1 && tableType ==
                                ↪ 3 {
334                                     vacancy.Course = "-"
335                             }
336                             fallthrough
337                     case 11:
338                             if rowIndex != 1 && tableType ==
                                ↪ 3 {
339                                     // Type of the vacancy
340                                     if len(bs) > 0 {
341                                             vacancy.Type =
                                                ↪ string(bs)
342                                     } // else if
                                        ↪ len(team.Vacancies) >
                                        ↪ 0 {
343                                     // If Type is empty, just
                                        ↪ get the correct type
344                                     // from the last register,
                                        ↪ if any
345                                     // vacancy.Type =
                                        ↪ team.Vacancies[len(team.Vacanci
346                                     // }
```

```
347                                         infoType = 12
348                                 }
349                         case 12:
350                                 if rowIndex != 1 && tableType ==
                                    ↪  3 && vacancy.Course != "-" {
351                                         vacancy.Course =
                                            ↪  string(bs)
352                                         infoType = 13
353                                         // If it's the discipline,
                                            ↪  just stop here!
354                                         break
355                                 }
356                                 // If it's not the discipline, we
                                    ↪  can continue to
357                                 // process the vacancies.. :D
358                                 fallthrough
359                         case 13:
360                                 if rowIndex != 1 && tableType ==
                                    ↪  3 {
361                                         vacancy.VacanciesNumber,
                                            ↪  err =
                                            ↪  intCache.Atoi(bs)
362                                         if err == nil {
363                                                 vacancy.Vacancies
                                                    ↪  =
                                                    ↪  stringCache.String(bs)
364                                         }
365                                         infoType = 14
366                                 }
367                         case 14:
368                                 if rowIndex != 1 && tableType ==
                                    ↪  3 {
369                                         _, err =
                                            ↪  intCache.Atoi(bs)
370                                         if err == nil {
371                                                 vacancy.Subscribed
                                                    ↪  =
                                                    ↪  stringCache.String(bs)
372                                         }
```

```
373                                     infoType = 15
374                             }
375                     case 15:
376                             if rowIndex != 1 && tableType ==
                                ↪  3 {
377                                     if bytes.Equal(bs,
                                        ↪  []byte("-")) {
378                                             bs = []byte("0")
379                                     } else {
380                                             _, err =
                                                ↪  intCache.Atoi(bs)
381                                     }
382                                     if err == nil {
383                                             vacancy.Pending =
                                                ↪  stringCache.String(bs)
384                                     }
385                                     infoType = 16
386                             }
387                     case 16:
388                             if rowIndex != 1 && tableType ==
                                ↪  3 {
389                                     vacancy.EnrolledNumber,
                                        ↪  err =
                                        ↪  intCache.Atoi(bs)
390                                     if err == nil {
391                                             vacancy.Enrolled
                                                ↪  =
                                                ↪  stringCache.String(bs)
392                                     }
393                                     infoType = 11
394                             }
395                     case 17:
396                             if rowIndex >= 3 && tableType ==
                                ↪  4 {
397                                     teacher :=
                                        ↪  format.Teacher{
398                                             Name:
                                                ↪  stringCache.String(bs),
399                                     }
```

```
400                                     if len(teacher.Name) > 0
                                        ↪ {
401                                             teacher.ID =
                                                ↪ hash(bs)
402                                             teachers[teacher]
                                                ↪ = struct{}{}
403                                             activity.Teachers
                                                ↪ =
                                                ↪ append(activity.Teachers,
                                                ↪ teacher.Name)
404                                     }
405                                     infoType = 18
406                             }
407                     case 18:
408                             if rowIndex >= 3 && tableType ==
                                ↪ 4 {
409                                     activity.Type =
                                        ↪ stringCache.String(bs)
410                                     infoType = 19
411                             }
412                     case 19:
413                             if rowIndex >= 3 && tableType ==
                                ↪ 4 {
414                                     activity.Workload =
                                        ↪ stringCache.String(bs)
415                                     infoType = 17
416                             }
417                     case 20:
418                             if tableType == 1 {
419                                     team.Tables[3].Rows =
                                        ↪ append(team.Tables[3].Rows,
                                        ↪ map[string]string{"key":
                                        ↪ "Observação", "value":
                                        ↪ stringCache.String(bs)})
420                                     tableType = 2
421                                     infoType = -1
422                             }
423                     }
424                     tmp.Reset()
```

```
425                         }
426                 } else if tagType == html.TextToken && columnIndex > -1 {
427                         tmp.Write(bytes.TrimSpace(doc.Text()))
428                 }
429         }
430         if errors.Cause(err) == io.EOF {
431                 err = nil
432         }
433         if err != nil {
434                 err = errors.WithMessage(err, spew.Sprintf(" - additional
                    ↪  context: discipline=%v - campus=%v", discipline,
                    ↪  campus))
435         }
436         return teams, err
437 }
```

Arquivo schedule.go

```
1  package usp2offers
2
3  import (
4          "errors"
5  )
6
7  var errScheduleEmpty = errors.New("schedule string cannot be empty")
```

Arquivo state.go

```
1  package usp2offers
2
3  import (
4          "encoding/gob"
5          "errors"
6
7          "github.com/fjorgemota/gurudamatricula/robot/format"
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9  )
10
11 var ErrInvalidStates = errors.New("usp2: Invalid states found (with
   ↪  len=0)")
```

```
12
13 type State struct {
14         Campus      format.Campus
15         Disciplines []format.DisciplineOffer
16         Count       int
17         Retries     int
18         Prefix      string
19         ID          string
20         Names       []string
21 }
22
23 type DisciplineState struct {
24         ID          string
25         Campus      format.Campus
26         Discipline  format.DisciplineOffer
27         IntCache    pool.AtoiCache
28         StringCache pool.StringCache
29         Teams       []*format.Team
30 }
31
32 func init() {
33         gob.Register(State{})
34         gob.Register([]State{})
35 }
```

B.1.19 Pasta tools/gqlgen

Arquivo main.go

```
1 // +build ignore
2
3 package main
4
5 import "github.com/99designs/gqlgen/cmd"
6
7 func main() {
8         cmd.Execute()
9 }
```

B.1.20 Pasta util/aehook

Arquivo hook.go

```
1  // +build appengine
2
3  package aehook
4
5  import (
6          "github.com/sirupsen/logrus"
7          "golang.org/x/net/context"
8          "google.golang.org/appengine/log"
9  )
10
11 type aeHook struct {
12         ctx context.Context
13 }
14
15 func (hook aeHook) Levels() []logrus.Level {
16         return logrus.AllLevels
17 }
18
19 func (hook aeHook) Fire(entry *logrus.Entry) error {
20         line, err := entry.String()
21         if err != nil {
22                 log.Errorf(hook.ctx, "Unable to read entry, %v", err)
23                 return err
24         }
25         switch entry.Level {
26         case logrus.PanicLevel:
27                 return log.Criticalf(hook.ctx, line)
28         case logrus.FatalLevel:
29                 return log.Criticalf(hook.ctx, line)
30         case logrus.ErrorLevel:
31                 return log.Errorf(hook.ctx, line)
32         case logrus.WarnLevel:
33                 return log.Warningf(hook.ctx, line)
34         case logrus.InfoLevel:
35                 return log.Infof(hook.ctx, line)
36         case logrus.DebugLevel:
```

```
37                  return log.Debugf(hook.ctx, line)
38          default:
39                  return nil
40          }
41  }
42
43  // NewAEHook returns a hook to the AppEngine logging system
44  func NewAEHook(ctx context.Context) logrus.Hook {
45          return aeHook{ctx}
46  }
```

Arquivo noop.go

```
1   // +build !appengine
2
3   package aehook
4
5   import (
6           "github.com/sirupsen/logrus"
7           "golang.org/x/net/context"
8   )
9
10  type noopHook struct{}
11
12  func (hook noopHook) Levels() []logrus.Level {
13          return []logrus.Level{}
14  }
15
16  func (hook noopHook) Fire(entry *logrus.Entry) error {
17          return nil
18  }
19
20  // NewAEHook returns a hook to the AppEngine logging system
21  func NewAEHook(_ context.Context) logrus.Hook {
22          return noopHook{}
23  }
```

B.1.21 Pasta util/cache

Arquivo appengine_cache.go

```
1  package cache
2
3  import (
4          "bytes"
5          "reflect"
6          "time"
7
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11
12         "google.golang.org/appengine/memcache"
13
14         "golang.org/x/net/context"
15 )
16
17 type appengineCache struct {
18         ctx     context.Context
19         coder   coder.Coder
20         timeout time.Duration
21 }
22
23 func (cache appengineCache) SetWithExpiration(key string, value
   ↪  interface{}, expiration time.Duration) error {
24         buf := pool.GetBytesBuffer()
25         defer pool.PutBytesBuffer(buf)
26         err := cache.coder.EncodeTo(value, buf)
27         if err == nil {
28                 memcacheItem := &memcache.Item{
29                         Key:        key,
30                         Value:      buf.Bytes(),
31                         Expiration: expiration,
32                 }
33                 err = memcache.Set(cache.ctx, memcacheItem)
34         }
35         if err != nil {
36                 err = errSet{err}
37         }
38         return err
```

```
39 }
40
41 func (cache appengineCache) Set(key string, value interface{}) error {
42         buf := pool.GetBytesBuffer()
43         defer pool.PutBytesBuffer(buf)
44         err := cache.coder.EncodeTo(value, buf)
45         if err == nil {
46                 memcacheItem := &memcache.Item{
47                         Key:        key,
48                         Value:      buf.Bytes(),
49                         Expiration: cache.timeout,
50                 }
51                 err = memcache.Set(cache.ctx, memcacheItem)
52         }
53         if err != nil {
54                 err = errSet{err}
55         }
56         return err
57 }
58
59 func (cache appengineCache) Get(key string, result interface{}) error {
60         memcacheItem, err := memcache.Get(cache.ctx, key)
61         if err == memcache.ErrCacheMiss {
62                 err = ErrCacheMiss
63         } else if err != nil {
64                 err = errGet{err}
65         }
66         if err == nil {
67                 reader := bytes.NewReader(memcacheItem.Value)
68                 err = cache.coder.DecodeFrom(reader, result)
69         }
70         return err
71 }
72
73 func (cache appengineCache) GetMulti(keys []string, results interface{})
   ↪  map[string]error {
74         resultv := reflect.ValueOf(results)
75         resultt := resultv.Type()
76         err := make(map[string]error, len(keys))
```

```
77          if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
            ↪  reflect.String {
78                  for _, key := range keys {
79                          err[key] = ErrInvalidResults
80                  }
81                  return err
82          }
83          var elemt reflect.Type
84          isPtr := false
85          elemt = resultt.Elem()
86          if elemt.Kind() == reflect.Ptr {
87                  elemt = elemt.Elem()
88                  isPtr = true
89          }
90          items, errGet := memcache.GetMulti(cache.ctx, keys)
91          if errGet == nil {
92                  for key, item := range items {
93                          elem := reflect.New(elemt)
94                          elemv := elem.Interface()
95                          err[key] =
                            ↪  cache.coder.DecodeFrom(bytes.NewReader(item.Value),
                            ↪  elemv)
96                          if err[key] == nil {
97                                  keyv := reflect.ValueOf(key)
98                                  if !isPtr {
99                                          elem = elem.Elem()
100                                 }
101                                 resultv.SetMapIndex(keyv, elem)
102                         }
103                 }
104                 for _, key := range keys {
105                         if _, ok := items[key]; ok {
106                                 continue
107                         }
108                         err[key] = ErrCacheMiss
109                 }
110         } else {
111                 for _, key := range keys {
112                         err[key] = errGet
```

```
113                 }
114         }
115         return err
116 }
117
118 func (cache appengineCache) SetMulti(keys []string, items interface{})
    ↪  map[string]error {
119         itemsv := reflect.ValueOf(items)
120         itemst := itemsv.Type()
121         err := make(map[string]error, len(keys))
122         if itemst.Kind() != reflect.Map || itemst.Key().Kind() !=
            ↪  reflect.String {
123                 for _, key := range keys {
124                         err[key] = ErrInvalidItems
125                 }
126                 return err
127         }
128         memcacheItems := make([]*memcache.Item, 0, len(keys))
129         for _, key := range keys {
130                 keyv := reflect.ValueOf(key)
131                 elemv := itemsv.MapIndex(keyv)
132                 if !elemv.IsValid() {
133                         err[key] = ErrInvalidItem
134                         continue
135                 }
136                 v := elemv.Interface()
137                 buf := pool.GetBytesBuffer()
138                 defer pool.PutBytesBuffer(buf)
139                 err[key] = cache.coder.EncodeTo(v, buf)
140                 if err[key] == nil {
141                         memcacheItems = append(memcacheItems,
                        ↪  &memcache.Item{
142                                 Key:        key,
143                                 Value:      buf.Bytes(),
144                                 Expiration: cache.timeout,
145                         })
146                 }
147         }
148         var errSet error
```

```
149         if len(memcacheItems) > 0 {
150                 errSet = memcache.SetMulti(cache.ctx, memcacheItems)
151                 for _, item := range memcacheItems {
152                         err[item.Key] = errSet
153                 }
154         }
155         return err
156 }
157
158 func (cache appengineCache) SetMultiWithExpiration(keys []string, items
    ↪  interface{}, expiration time.Duration) map[string]error {
159         itemsv := reflect.ValueOf(items)
160         itemst := itemsv.Type()
161         err := make(map[string]error, len(keys))
162         if itemst.Kind() != reflect.Map || itemst.Key().Kind() !=
          ↪  reflect.String {
163                 for _, key := range keys {
164                         err[key] = ErrInvalidItems
165                 }
166                 return err
167         }
168         memcacheItems := make([]*memcache.Item, 0, len(keys))
169         for _, key := range keys {
170                 keyv := reflect.ValueOf(key)
171                 elemv := itemsv.MapIndex(keyv)
172                 if !elemv.IsValid() {
173                         err[key] = ErrInvalidItem
174                         continue
175                 }
176                 v := elemv.Interface()
177                 buf := pool.GetBytesBuffer()
178                 defer pool.PutBytesBuffer(buf)
179                 err[key] = cache.coder.EncodeTo(v, buf)
180                 if err[key] == nil {
181                         memcacheItems = append(memcacheItems,
                          ↪  &memcache.Item{
182                                 Key:        key,
183                                 Value:      buf.Bytes(),
184                                 Expiration: expiration,
```

```
185                         })
186                 }
187         }
188         var errSet error
189         if len(memcacheItems) > 0 {
190                 errSet = memcache.SetMulti(cache.ctx, memcacheItems)
191                 for _, item := range memcacheItems {
192                         err[item.Key] = errSet
193                 }
194         }
195         return err
196 }
197 func (cache appengineCache) Pop(key string, result interface{}) error {
198         err := cache.Get(key, result)
199         if err == nil {
200                 err = cache.Del(key)
201         }
202         return err
203 }
204
205 func (cache appengineCache) Del(key string) error {
206         err := memcache.Delete(cache.ctx, key)
207         if err == memcache.ErrCacheMiss {
208                 err = nil // If the key does not exist, we can ignore
                    ↪  it..
209         }
210         return err
211 }
212
213 // NewAppEngineCache returns an Cache that saves data on Memcache, so
    ↪  it's
214 // available to any instance that can access Memcache
215 func NewAppEngineCache(ctx context.Context, coder coder.Coder, timeout
    ↪  time.Duration) Cache {
216         return appengineCache{
217                 ctx:     ctx,
218                 timeout: timeout,
219                 coder:   coder,
220         }
```

```
}
```

Arquivo base.go

```
package cache

import (
        "errors"
        "time"
)

var (
        // ErrCacheMiss is an error returned only when the key actually
        ↪  does not
        // exist in the cache
        ErrCacheMiss = errGet{errors.New("cache: cache miss")}

        // ErrInvalidResults is returned when results is not a map in the
        ↪  expected
        // format
        ErrInvalidResults = errGet{errors.New("cache: results must be a
        ↪  map with key as a string")}

        // ErrInvalidItems is returned when results is not a map in the
        ↪  expected
        // format
        ErrInvalidItems = errSet{errors.New("cache: items must be a map
        ↪  with key as a string")}

        // ErrInvalidItem is returned when a key in the items is a zero
        ↪  value
        ErrInvalidItem = errSet{errors.New("cache: the item must not be a
        ↪  zero value")}
)

// item is a generic cache item with value and expiration information
type item struct {
        Value    []byte `json:"value"`
        ExpireAt int64  `json:"expire_at"`
}
```

```
30
31 /*
32 Cache implements a basic caching interface
33 */
34 type Cache interface {
35         SetWithExpiration(key string, value interface{}, expiration
           ↪  time.Duration) error
36         Set(key string, value interface{}) error
37         Get(key string, result interface{}) error
38         Pop(key string, result interface{}) error
39         GetMulti(keys []string, results interface{}) map[string]error
40         SetMulti(keys []string, items interface{}) map[string]error
41         SetMultiWithExpiration(keys []string, items interface{},
           ↪  expiration time.Duration) map[string]error
42         Del(key string) error
43 }
```

Arquivo err_get.go

```
1  package cache
2
3  type errGet struct {
4          err error
5  }
6
7  func (eg errGet) Error() string {
8          return eg.err.Error()
9  }
10
11 // IsGetError returns true if the error was caused by a Get, and not by a
   ↪  Set,
12 // or false otherwise
13 func IsGetError(err error) bool {
14         _, ok := err.(errGet)
15         return ok
16 }
```

Arquivo err_set.go

```go
package cache

type errSet struct {
        err error
}

func (eg errSet) Error() string {
        return eg.err.Error()
}

// IsSetError returns true if the error was caused by a Set, and not by a
↪  Get,
// or false otherwise
func IsSetError(err error) bool {
        _, ok := err.(errSet)
        return ok
}
```

Arquivo file_cache.go

```go
package cache

import (
        "bytes"
        "io"
        "time"

        "github.com/fjorgemota/gurudamatricula/util/clock"
        "github.com/fjorgemota/gurudamatricula/util/coder"
        "github.com/fjorgemota/gurudamatricula/util/file"
        "github.com/fjorgemota/gurudamatricula/util/pool"
)

type fileCache struct {
        coder   coder.Coder
        parent  Cache
        manager file.Manager
        timeout time.Duration
        clk     clock.Clock
}
```

```
21
22 func (cache fileCache) setWithExpiration(key string, value interface{},
   ↪  expiration time.Duration) error {
23         var expireAt int64
24         if expiration == 0 {
25                 expireAt = 0
26         } else {
27                 expireAt = cache.clk.Now().Add(expiration).Unix()
28         }
29         buf := pool.GetBytesBuffer()
30         defer pool.PutBytesBuffer(buf)
31         err := cache.coder.EncodeTo(value, buf)
32         gcsItem := item{
33                 Value:    buf.Bytes(),
34                 ExpireAt: expireAt,
35         }
36         var writer io.WriteCloser
37         if err == nil {
38                 writer, err = cache.manager.Writer(key)
39         }
40         if err == nil {
41                 err = cache.coder.EncodeTo(gcsItem, writer)
42         }
43         if err == nil {
44                 err = writer.Close()
45         }
46         if err != nil {
47                 err = errSet{err}
48         }
49         return err
50 }
51
52 func (cache fileCache) SetWithExpiration(key string, value interface{},
   ↪  expiration time.Duration) error {
53         return cache.setWithExpiration(key, value, expiration)
54 }
55
56 func (cache fileCache) set(key string, value interface{}) error {
57         var expireAt int64
```

```
58          if cache.timeout == 0 {
59                  expireAt = 0
60          } else {
61                  expireAt = cache.clk.Now().Add(cache.timeout).Unix()
62          }
63          buf := pool.GetBytesBuffer()
64          defer pool.PutBytesBuffer(buf)
65          err := cache.coder.EncodeTo(value, buf)
66          gcsItem := item{
67                  Value:    buf.Bytes(),
68                  ExpireAt: expireAt,
69          }
70          var writer io.WriteCloser
71          if err == nil {
72                  writer, err = cache.manager.Writer(key)
73          }
74          if err == nil {
75                  err = cache.coder.EncodeTo(gcsItem, writer)
76          }
77          if err == nil {
78                  err = writer.Close()
79          }
80          if err != nil {
81                  err = errSet{err}
82          }
83          return err
84  }
85
86  func (cache fileCache) Set(key string, value interface{}) error {
87          return cache.set(key, value)
88  }
89
90  func (cache fileCache) get(key string, result interface{}) error {
91          reader, err := cache.manager.Reader(key)
92          var cacheItem item
93          if err == nil {
94                  // We will buffer the bytes of the file so we can work
                    ↪  with it
95                  if err == nil {
```

```
96                      err = cache.coder.DecodeFrom(reader, &cacheItem)
97              }
98              if err == nil {
99                      err = reader.Close()
100             }
101             if err == nil &&
102                     cacheItem.ExpireAt != 0 &&
103                     cacheItem.ExpireAt < cache.clk.Now().Unix() {
104                     err = ErrCacheMiss
105             }
106             if err == nil {
107                     // Decodify value..
108                     reader := bytes.NewReader(cacheItem.Value)
109                     err = cache.coder.DecodeFrom(reader, result)
110             }
111     } else if file.IsNotExistError(err) {
112             err = ErrCacheMiss
113     }
114     if err != nil && err != ErrCacheMiss {
115             err = errGet{err}
116     }
117     return err
118 }
119
120 func (cache fileCache) Get(key string, result interface{}) error {
121     return cache.get(key, result)
122 }
123
124 func (cache fileCache) GetMulti(keys []string, results interface{})
    ↪  map[string]error {
125     return getMulti(cache.get, cache, keys, results)
126 }
127
128 func (cache fileCache) SetMulti(keys []string, items interface{})
    ↪  map[string]error {
129     return setMulti(cache.set, keys, items)
130 }
131
```

```
132 func (cache fileCache) SetMultiWithExpiration(keys []string, items
    ↪  interface{}, expiration time.Duration) map[string]error {
133         return setMultiWithExpiration(cache.setWithExpiration, keys,
            ↪  items, expiration)
134 }
135
136 func (cache fileCache) Pop(key string, result interface{}) error {
137         err := cache.Get(key, result)
138         if err == nil {
139                 err = cache.Del(key)
140         }
141         return err
142 }
143
144 func (cache fileCache) Del(key string) error {
145         err := cache.manager.Delete(key)
146         if file.IsNotExistError(err) {
147                 err = nil // If the file does not exist, we can ignore
                    ↪  it..
148         }
149         return err
150 }
151
152 // NewFileCache returns a cache that is constantly saved on a file
    ↪  storage for
153 // persistence. So...any data that is saved here is not lost when the
    ↪  instance
154 // is terminated, for example
155 func NewFileCache(
156         manager file.Manager,
157         coder coder.Coder,
158         timeout time.Duration,
159         clk clock.Clock,
160 ) Cache {
161         coder.Register(item{})
162         return fileCache{
163                 clk:     clk,
164                 coder:   coder,
165                 manager: manager,
```

```
166                 timeout: timeout,
167         }
168 }
```

Arquivo get_multi.go

```
1  package cache
2
3  import (
4          "reflect"
5  )
6
7  func getMulti(get func(string, interface{}) error, cache Cache, keys
   ↪  []string, results interface{}) map[string]error {
8          resultv := reflect.ValueOf(results)
9          resultt := resultv.Type()
10         err := make(map[string]error, len(keys))
11         if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
           ↪  reflect.String {
12                 for _, key := range keys {
13                         err[key] = ErrInvalidResults
14                 }
15                 return err
16         }
17         var elemt reflect.Type
18         isPtr := false
19         elemt = resultt.Elem()
20         if elemt.Kind() == reflect.Ptr {
21                 elemt = elemt.Elem()
22                 isPtr = true
23         }
24         var errKeys []string
25         for _, key := range keys {
26                 elemv := reflect.New(elemt)
27                 v := elemv.Interface()
28                 err[key] = get(key, v)
29                 if err[key] == nil {
30                         keyv := reflect.ValueOf(key)
31                         elemv = reflect.ValueOf(v)
32                         if !isPtr {
```

```
33                                      elemv = elemv.Elem()
34                              }
35                              resultv.SetMapIndex(keyv, elemv)
36                      } else {
37                              errKeys = append(errKeys, key)
38                      }
39              }
40              return err
41      }
```

Arquivo lru_cache.go

```
1   package cache
2
3   import (
4           "bytes"
5           "time"
6
7           "github.com/fjorgemota/gurudamatricula/util/clock"
8           "github.com/fjorgemota/gurudamatricula/util/coder"
9           lru "github.com/hashicorp/golang-lru"
10  )
11
12  /*
13  lruCache is a cache available only locally to the machine acessing it. It
    ↪  is not
14  available remotely and access the data directly from the memory of the
    ↪  local
15  machine. It's definitely the fastest cache available in the world.
16  */
17  type lruCache struct {
18          lru     *lru.Cache
19          parent  Cache
20          timeout time.Duration
21          coder   coder.Coder
22          clk     clock.Clock
23  }
24
25  func (cache lruCache) setWithExpiration(key string, value interface{},
    ↪  expiration time.Duration) error {
```

```
26          var expireAt int64
27          if expiration == 0 {
28                  expireAt = 0
29          } else {
30                  expireAt = cache.clk.Now().Add(expiration).Unix()
31          }
32          var buf bytes.Buffer
33          err := cache.coder.EncodeTo(value, &buf)
34          if err == nil {
35                  lruItem := item{
36                          Value:    buf.Bytes(),
37                          ExpireAt: expireAt,
38                  }
39                  cache.lru.Add(key, lruItem)
40          } else {
41                  err = errSet{err}
42          }
43          return err
44  }
45
46  /*
47  SetWithExpiration saves a key and a value in the LRU cache, so we can
    ↪  access it
48   faster.
49   The difference between this method and Set() is that, with this method,
    ↪  we can
50   set a specific expiration for the key in question. So the data will not
    ↪  be
51   available after `expiration` seconds.
52  */
53  func (cache lruCache) SetWithExpiration(
54          key string,
55          value interface{},
56          expiration time.Duration) error {
57          return cache.setWithExpiration(key, value, expiration)
58  }
59
60  func (cache lruCache) set(key string, value interface{}) error {
61          var expireAt int64
```

```
62          if cache.timeout == 0 {
63                  expireAt = 0
64          } else {
65                  expireAt = cache.clk.Now().Add(cache.timeout).Unix()
66          }
67          var buf bytes.Buffer
68          err := cache.coder.EncodeTo(value, &buf)
69          if err == nil {
70                  lruItem := item{
71                          Value:    buf.Bytes(),
72                          ExpireAt: expireAt,
73                  }
74                  cache.lru.Add(key, lruItem)
75          } else {
76                  err = errSet{err}
77          }
78          return err
79  }
80
81  /*
82  Set saves a key and a value in the LRU cache, so we can access it faster.
83  Note that this method will set the same key and value in the parent's
    ↪  cache,
84  so any code using Get() with the same key which does not have it in it's
    ↪  LRU
85  cache will be able to access it from a parent cache, such as Memcache,
    ↪  for
86  example.
87  */
88  func (cache lruCache) Set(key string, value interface{}) error {
89          return cache.set(key, value)
90  }
91
92  func (cache lruCache) get(key string, result interface{}) error {
93          lruItem, ok := cache.lru.Get(key)
94          var cacheItem item
95          if ok {
96                  cacheItem, ok = lruItem.(item)
97          }
```

```
98          if cacheItem.ExpireAt != 0 && cacheItem.ExpireAt <
            ↪  cache.clk.Now().Unix() {
99                  ok = false
100         }
101         var err error
102         if ok {
103                 err =
                    ↪  cache.coder.DecodeFrom(bytes.NewReader(cacheItem.Value),
                    ↪  result)
104                 if err != nil {
105                         err = errGet{err}
106                 }
107         } else {
108                 err = ErrCacheMiss
109         }
110         return err
111 }
112
113 /*
114 Get gets a key from the LRU cache and, if it yet does not exists, calls
115 the parent cache. If the key exists in the parent cache, we'll save it's
    ↪  value
116 to the LRU cache, so we can access it faster.
117 */
118 func (cache lruCache) Get(key string, result interface{}) error {
119         return cache.get(key, result)
120 }
121
122 /*
123 GetMUlti get many keys at once and put it into a map
124 */
125 func (cache lruCache) GetMulti(keys []string, results interface{})
    ↪  map[string]error {
126         return getMulti(cache.get, cache, keys, results)
127 }
128
129 /*
130 SetMulti sets many keys at once
131 */
```

```
132 func (cache lruCache) SetMulti(keys []string, items interface{})
    ↪ map[string]error {
133         return setMulti(cache.set, keys, items)
134 }
135
136 /*
137 SetMUlti sets many keys at once with a custom expiration
138 */
139 func (cache lruCache) SetMultiWithExpiration(keys []string, items
    ↪ interface{}, expiration time.Duration) map[string]error {
140         return setMultiWithExpiration(cache.setWithExpiration, keys,
            ↪ items, expiration)
141 }
142
143 /*
144 Pop will get the key and delete it from the cache, including from the
    ↪ parent
145 cache
146 */
147 func (cache lruCache) Pop(key string, result interface{}) error {
148         err := cache.Get(key, result)
149         if err == nil {
150                 err = cache.Del(key)
151         }
152         return err
153 }
154
155 /*
156 Del will remove the key from the LRU and from the parent cache.
157 */
158 func (cache lruCache) Del(key string) error {
159         cache.lru.Remove(key)
160         return nil
161 }
162
163 /*
164 NewLRUCache will return a new LRU cache with the informed capacity,
    ↪ default
```

```
165 timeout and parent cache, which saves information just in this instance's
    ↪ memory
166 */
167 func NewLRUCache(
168         clk clock.Clock,
169         cod coder.Coder,
170         capacity int,
171         timeout time.Duration) (Cache, error) {
172         cache, err := lru.New(capacity)
173         if err != nil {
174                 return nil, err
175         }
176         instance := lruCache{
177                 lru:     cache,
178                 timeout: timeout,
179                 coder:   cod,
180                 clk:     clk,
181         }
182         return instance, nil
183 }
```

Arquivo multi_cache.go

```
1  package cache
2
3  import (
4          "time"
5  )
6
7  type multiCache struct {
8          children []Cache
9  }
10
11 func (mc multiCache) SetWithExpiration(key string, value interface{},
   ↪ expiration time.Duration) error {
12         var err error
13         for _, child := range mc.children {
14                 if err == nil {
15                         err = child.SetWithExpiration(key, value,
                           ↪ expiration)
```

```
16                  }
17          }
18          return err
19  }
20
21  func (mc multiCache) Set(key string, value interface{}) error {
22          var err error
23          for _, child := range mc.children {
24                  if err == nil {
25                          err = child.Set(key, value)
26                  }
27          }
28          return err
29  }
30
31  func (mc multiCache) Get(key string, result interface{}) error {
32          var err error
33          for i, child := range mc.children {
34                  if i == 0 || err != nil {
35                          err = child.Get(key, result)
36                  }
37                  if !IsGetError(err) {
38                          for _, newChild := range mc.children[:i] {
39                                  if err == nil {
40                                          err = newChild.Set(key, result)
41                                  }
42                          }
43                          break
44                  }
45          }
46          return err
47  }
48
49  func (mc multiCache) Pop(key string, result interface{}) error {
50          var err error
51          for i, child := range mc.children {
52                  if i == 0 || err != nil {
53                          err = child.Pop(key, result)
54                  }
```

```
55                  if err == nil {
56                          for _, newChild := range mc.children[i:] {
57                                  if err == nil {
58                                          err = newChild.Del(key)
59                                  }
60                          }
61                          break
62                  }
63          }
64          return err
65  }
66
67  func (mc multiCache) GetMulti(keys []string, results interface{})
    ↪  map[string]error {
68          err := make(map[string]error, len(keys))
69          for _, key := range keys {
70                  err[key] = ErrCacheMiss
71          }
72          for i, child := range mc.children {
73                  keys = keys[:0]
74                  for key, value := range err {
75                          if IsGetError(value) {
76                                  keys = append(keys, key)
77                          }
78                  }
79                  if len(keys) > 0 {
80                          errChild := child.GetMulti(keys, results)
81                          keys = keys[:0]
82                          for key, value := range errChild {
83                                  err[key] = value
84                                  if !IsGetError(value) {
85                                          keys = append(keys, key)
86                                  }
87                          }
88                          if len(keys) > 0 {
89                                  // If any key succeeded, use SetMulti to
                                    ↪  save the keys from the
90                                  // parent to the actual cache instance
```

```
91                                      for _, newChild := range mc.children[:i]
                                        ↪ {
92                                              errParent :=
                                                ↪ newChild.SetMulti(keys,
                                                ↪ results)
93                                              for key, value := range errParent
                                                ↪ {
94                                                      err[key] = value
95                                              }
96                                      }
97                              }
98                      }
99
100             }
101             return err
102     }
103
104     func (mc multiCache) SetMulti(keys []string, items interface{})
        ↪ map[string]error {
105             err := make(map[string]error, len(keys))
106             for _, key := range keys {
107                     err[key] = nil
108             }
109             for _, child := range mc.children {
110                     keys = keys[:0]
111                     for key, value := range err {
112                             if value == nil {
113                                     keys = append(keys, key)
114                             }
115                     }
116                     if len(keys) > 0 {
117                             errChild := child.SetMulti(keys, items)
118                             for key, value := range errChild {
119                                     err[key] = value
120                             }
121                     }
122             }
123             return err
124     }
```

```
125
126 func (mc multiCache) SetMultiWithExpiration(keys []string, items
    ↪  interface{}, expiration time.Duration) map[string]error {
127         err := make(map[string]error, len(keys))
128         for _, key := range keys {
129                 err[key] = nil
130         }
131         for _, child := range mc.children {
132                 keys = keys[:0]
133                 for key, value := range err {
134                         if value == nil {
135                                 keys = append(keys, key)
136                         }
137                 }
138                 if len(keys) > 0 {
139                         errChild := child.SetMultiWithExpiration(keys,
                        ↪  items, expiration)
140                         for key, value := range errChild {
141                                 err[key] = value
142                         }
143                 }
144         }
145         return err
146 }
147
148 func (mc multiCache) Del(key string) error {
149         var err error
150         for _, child := range mc.children {
151                 if err == nil {
152                         err = child.Del(key)
153                 }
154         }
155         return err
156 }
157
158 func NewMultiCache(children ...Cache) Cache {
159         return multiCache{
160                 children: children,
161         }
```

```
}
```

Arquivo nil_cache.go

```
package cache

import "time"

// NilCache implements a struct that is just a no-op implementation of the
// Cache interface
type nilCache struct{}

// SetWithExpiration implements a no-op method that just returns nil for all the
// queries
func (cache nilCache) SetWithExpiration(key string, value interface{}, expiration time.Duration) error {
        return nil
}

// Set implements a no-op method that just returns nil for all the queries
func (cache nilCache) Set(key string, value interface{}) error {
        return nil
}

// Get implements a no-op method that just returns ErrCacheMiss error for all
// the queries
func (cache nilCache) Get(key string, result interface{}) error {
        return ErrCacheMiss
}

// GetMulti implements a no-op method that just returns ErrCacheMiss for
// all the queries
func (cache nilCache) GetMulti(keys []string, result interface{}) map[string]error {
        err := make(map[string]error, len(keys))
        for _, key := range keys {
```

```
31                  err[key] = ErrCacheMiss
32          }
33          return err
34  }
35
36  // SetMulti implements a no-op method that just returns nil for
37  // all the queries
38  func (cache nilCache) SetMulti(keys []string, _ interface{})
    ↪  map[string]error {
39          return cache.SetMultiWithExpiration(keys, nil, 0*time.Second)
40  }
41
42  // SetMultiWithExpiration implements a no-op method that just returns nil
    ↪  for
43  // all the queries
44  func (cache nilCache) SetMultiWithExpiration(keys []string, _ interface{},
    ↪  _ time.Duration) map[string]error {
45          err := make(map[string]error, len(keys))
46          for _, key := range keys {
47                  err[key] = nil
48          }
49          return err
50  }
51
52  // Pop implements a no-op method that just returns ErrCacheMiss error for
    ↪  all
53  // the queries
54  func (cache nilCache) Pop(key string, result interface{}) error {
55          return ErrCacheMiss
56  }
57
58  // Del implements a no-op method that just returns nil
59  func (cache nilCache) Del(key string) error {
60          return nil
61  }
62
63  // NewNilCache creates a new NilCache instance, that acts like an
    ↪  /dev/null for
64  // cache instances
```

```
65 func NewNilCache() Cache {
66         return nilCache{}
67 }
```

Arquivo set_multi.go

```
1  package cache
2
3  import (
4          "reflect"
5          "time"
6  )
7
8  func setMulti(set func(string, interface{}) error, keys []string, items
   ↪  interface{}) map[string]error {
9          itemsv := reflect.ValueOf(items)
10         itemst := itemsv.Type()
11         err := make(map[string]error, len(keys))
12         if itemst.Kind() != reflect.Map || itemst.Key().Kind() !=
           ↪  reflect.String {
13                 for _, key := range keys {
14                         err[key] = ErrInvalidItems
15                 }
16                 return err
17         }
18         var sucKeys []string
19         for _, key := range keys {
20                 keyv := reflect.ValueOf(key)
21                 elemv := itemsv.MapIndex(keyv)
22                 if !elemv.IsValid() {
23                         err[key] = ErrInvalidItem
24                         continue
25                 }
26                 v := elemv.Interface()
27                 err[key] = set(key, v)
28                 if err[key] == nil {
29                         sucKeys = append(sucKeys, key)
30                 }
31         }
32         return err
```

```
}

func setMultiWithExpiration(set func(string, interface{}, time.Duration)
↪  error, keys []string, items interface{}, expiration time.Duration)
↪  map[string]error {
        itemsv := reflect.ValueOf(items)
        itemst := itemsv.Type()
        err := make(map[string]error, len(keys))
        if itemst.Kind() != reflect.Map || itemst.Key().Kind() !=
        ↪  reflect.String {
                for _, key := range keys {
                        err[key] = ErrInvalidItems
                }
                return err
        }
        var sucKeys []string
        for _, key := range keys {
                keyv := reflect.ValueOf(key)
                elemv := itemsv.MapIndex(keyv)
                if !elemv.IsValid() {
                        err[key] = ErrInvalidItem
                        continue
                }
                v := elemv.Interface()
                err[key] = set(key, v, expiration)
                if err[key] == nil {
                        sucKeys = append(sucKeys, key)
                }
        }
        return err
}
```

B.1.22 Pasta util/clock

Arquivo base.go

```
package clock

import "time"

```

```
5  // Clock is a interface defining a simple Clock
6  type Clock interface {
7          Now() time.Time
8          Sleep(d time.Duration)
9  }
```

Arquivo fake.go

```
1  package clock
2
3  import "time"
4
5  // fakeClock simulates a fake clock
6  type fakeClock struct {
7          slept time.Duration
8  }
9
10 // Now returns actual time added by slept
11 func (f *fakeClock) Now() time.Time {
12         return time.Now().Add(f.slept)
13 }
14
15 // Sleep sleeps clock for d duration
16 func (f *fakeClock) Sleep(d time.Duration) {
17         f.slept += d
18 }
19
20 // NewFakeClock returns a new fake clock
21 func NewFakeClock() Clock {
22         return &fakeClock{}
23 }
```

Arquivo real.go

```
1  package clock
2
3  import "time"
4
5  // realClock implements a fake clock
```

```
type realClock struct{}

// Now returns actual time added by slept
func (f realClock) Now() time.Time {
        return time.Now()
}

// Sleep sleeps clock for d duration
func (f realClock) Sleep(d time.Duration) {
        time.Sleep(d)
}

// NewRealClock returns a new real clock
func NewRealClock() Clock {
        return realClock{}
}
```

B.1.23 Pasta util/coder

Arquivo base.go

```
package coder

import (
        "errors"
        "io"
)

var (
        // ErrNotPointer is an error returned when the passed argument is
        ↪  not a pointer
        errNotPointer = errors.New("coder: argument must be a pointer")
        // ErrNotStruct is an error returned when the passed argument is
        ↪  not a struct
        errNotStruct = errors.New("coder: argument must be a struct")
        // ErrAlreadyClosed is returned when a streaming encoder is
        ↪  closed and
        // the user tried to write to it
        errAlreadyClosed = errors.New("coder: the encoder is already
        ↪  closed")
```

```
16          // ErrUnexpectedByte is returned when a streaming decoder founds
            ↪  a byte
17          // that is totally unexpected
18          errUnexpectedByte = errors.New("coder: unexpected byte")
19  )
20
21  // Encoder is the interface describing data that provides its own
    ↪  representation
22  // for encoding values for transmission to a Decoder.
23  type Encoder interface {
24          EncodeTo(coder Coder, target io.Writer) error
25  }
26
27  // Decoder is the interface describing data that provides its own routine
    ↪  for
28  // decoding transmitted values sent by a Encoder.
29  type Decoder interface {
30          DecodeFrom(coder Coder, source io.Reader) error
31  }
32
33  /*
34  Coder is a interface that implement an encoder/decoder, so we can easily
    ↪  save
35  information to Memcached and Google Cloud Storage, for example
36  */
37  type Coder interface {
38          Register(val interface{})
39          DecodeFrom(source io.Reader, result interface{}) error
40          EncodeTo(result interface{}, target io.Writer) error
41  }
42
43  // StreamingEncoder encodes and writes to an io.Writer, item by item,
    ↪  different
44  // objects
45  type StreamingEncoder interface {
46          Encode(val interface{}) error
47          Flush() error
48          Close() error
49  }
```

```
50
51 // StreamingDecoder decodes a encoded io.Reader, item by item, into
   ↪  diferent
52 // objects
53 type StreamingDecoder interface {
54         Decode(val interface{}) error
55         Ignore() error
56 }
57
58 // IntEncoder encodes numbers to string, so they seems to be random yet
   ↪  it
59 // is deterministic
60 type IntEncoder interface {
61         Encode(n int) (string, error)
62         EncodeInt64(n int64) (string, error)
63 }
```

Arquivo bzip2.go

```
1  package coder
2
3  import (
4          "io"
5
6          "github.com/dsnet/compress/bzip2"
7  )
8
9  type bzip2Handle struct {
10         level int
11 }
12
13 func (zh bzip2Handle) Writer(w io.Writer) (io.WriteCloser, error) {
14         return bzip2.NewWriter(w, &bzip2.WriterConfig{
15                 Level: zh.level,
16         })
17 }
18 func (zh bzip2Handle) Reader(r io.Reader) (io.ReadCloser, error) {
19         return bzip2.NewReader(r, nil)
20 }
21
```

```
22 func NewBZIP2Handle(level int) CompressorHandle {
23         return bzip2Handle{level: level}
24 }
```

Arquivo compressor_coder.go

```
1  package coder
2
3  import (
4          "bytes"
5          "io"
6          "reflect"
7
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9
10         "github.com/pkg/errors"
11 )
12
13 // CompressorHandle defines an interface to support various compressors
14 type CompressorHandle interface {
15         Writer(io.Writer) (io.WriteCloser, error)
16         Reader(io.Reader) (io.ReadCloser, error)
17 }
18
19 /*
20 compressorCoder creates an encoder that calls a parent Coder, verifies if
   ↪  it's
21 size is acceptable (e.g is below the threshold of the compressor) and, if
   ↪  it's
22 not, compress the output emitted by the parent Coder
23 */
24 type compressorCoder struct {
25         parent    Coder
26         handle    CompressorHandle
27         threshold int
28 }
29
30 /*
31 Register registers a new type in a parent coder, if necessary
32 */
```

```
func (cod compressorCoder) Register(val interface{}) {
        cod.parent.Register(val)
}

type compressorWrapper struct {
        target     io.Writer
        buf        *bytes.Buffer
        compressor io.WriteCloser
        parent     compressorCoder
}

func (cw *compressorWrapper) Write(bs []byte) (int, error) {
        var n int
        var err error
        if cw.buf != nil {
                n, err = cw.buf.Write(bs)
                if err == nil && cw.buf.Len() > cw.parent.threshold {
                        cw.compressor, err =
                        ↪  cw.parent.handle.Writer(cw.target)
                        if err == nil {
                                _, err = io.Copy(cw.compressor, cw.buf)
                        }
                        pool.PutBytesBuffer(cw.buf)
                        cw.buf = nil
                }
        } else {
                n, err = cw.compressor.Write(bs)
        }
        return n, err
}

func (cw *compressorWrapper) Close() error {
        var err error
        if cw.buf != nil {
                _, err = io.Copy(cw.target, cw.buf)
                pool.PutBytesBuffer(cw.buf)
                cw.buf = nil
        } else if cw.compressor != nil {
                err = cw.compressor.Close()
```

```
71          }
72          return err
73  }
74
75  func (cod compressorCoder) EncodeTo(val interface{}, target io.Writer)
    ↪  error {
76          if e, ok := val.(Encoder); ok {
77                  return e.EncodeTo(cod, target)
78          }
79          writer := &compressorWrapper{
80                  buf:    pool.GetBytesBuffer(),
81                  parent: cod,
82                  target: target,
83          }
84          err := cod.parent.EncodeTo(val, writer)
85          if err == nil {
86                  err = writer.Close()
87          }
88          return err
89  }
90
91  func (cod compressorCoder) DecodeFrom(source io.Reader, result
    ↪  interface{}) error {
92          if reflect.ValueOf(result).Kind() != reflect.Ptr {
93                  return errors.WithStack(errNotPointer)
94          }
95          if d, ok := result.(Decoder); ok {
96                  return d.DecodeFrom(cod, source)
97          }
98          buf := pool.GetBytesBuffer()
99          defer pool.PutBytesBuffer(buf)
100         reader := io.TeeReader(source, buf)
101         descompressor, err := cod.handle.Reader(reader)
102         if err == nil {
103                 err = cod.parent.DecodeFrom(descompressor, result)
104         }
105         if err == nil {
106                 err = descompressor.Close()
107         }
```

```
108         if err != nil {
109                 err = cod.parent.DecodeFrom(io.MultiReader(buf, reader),
                    ↪  result)
110         }
111         return err
112 }
113
114 /*
115 NewCompressorCoder returns a customized new CompressorCoder
116 */
117 func NewCompressorCoder(parent Coder, handle CompressorHandle, threshold
    ↪  int) Coder {
118         return compressorCoder{
119                 handle:    handle,
120                 threshold: threshold,
121                 parent:    parent,
122         }
123 }
```

Arquivo csv_decoder.go

```
1  package coder
2
3  import (
4          "bufio"
5          "encoding/csv"
6          "io"
7          "reflect"
8
9          "github.com/fjorgemota/gurudamatricula/util/pool"
10         "github.com/pkg/errors"
11 )
12
13 type csvDecoder struct {
14         types  map[reflect.Type]map[int]int
15         header map[string]int
16         reader *csv.Reader
17         buf    *bufio.Reader
18 }
19
```

```
20 func (cp *csvDecoder) parse() error {
21         var err error
22         if cp.header == nil {
23                 var header []string
24                 header, err = cp.read()
25                 mapping := make(map[string]int, len(header))
26                 for c, head := range header {
27                         mapping[head] = c
28                 }
29                 cp.header = mapping
30         }
31         return err
32 }
33
34 func (cp *csvDecoder) recognize(typ reflect.Type) map[int]int {
35         if mapping, ok := cp.types[typ]; ok {
36                 return mapping
37         }
38         mapping := make(map[int]int)
39         fields := typ.NumField()
40         for c := 0; c < fields; c++ {
41                 field := typ.Field(c)
42                 tag := field.Tag.Get("csv")
43                 if tag == "" {
44                         // Ignore field without a tag
45                         continue
46                 }
47                 headPos, ok := cp.header[tag]
48                 if !ok {
49                         // Ignore field that does not exist in the CSV
50                         continue
51                 }
52                 mapping[headPos] = c
53         }
54         cp.types[typ] = mapping
55         return mapping
56 }
57 func (cp *csvDecoder) read() ([]string, error) {
58         var row []string
```

```
59          err := io.EOF
60          if cp.reader != nil {
61                  row, err = cp.reader.Read()
62          }
63          if cp.reader != nil && err == io.EOF {
64                  pool.PutBufioReader(cp.buf)
65                  cp.reader = nil
66                  cp.buf = nil
67          }
68          return row, err
69  }
70
71  func (cp *csvDecoder) Decode(result interface{}) error {
72          var err error
73          if cp.header == nil {
74                  err = cp.parse()
75          }
76          var row []string
77          if err == nil {
78                  row, err = cp.read()
79          }
80          var value reflect.Value
81          if err == nil {
82                  value, err = validateType(result)
83          }
84          if err == nil {
85                  resultv := value.Elem()
86                  mapping := cp.recognize(resultv.Type())
87                  for c, column := range row {
88                          if field, ok := mapping[c]; ok {
89                                  resultv.Field(field).SetString(column)
90                          }
91                  }
92          }
93          return errors.WithStack(err)
94  }
95
96  func (cp *csvDecoder) Ignore() error {
97          var err error
```

```
98          if cp.header == nil {
99                  err = cp.parse()
100         }
101         if err == nil {
102                 _, err = cp.reader.Read()
103         }
104         return err
105 }
106
107 // NewCSVDecoder returns a iterator that is able to process the CSV into
108 // structs based on it's tag
109 func NewCSVDecoder(reader io.Reader) StreamingDecoder {
110         types := make(map[reflect.Type]map[int]int)
111         buf := pool.GetBufioReader(reader)
112         parser := csv.NewReader(buf)
113         parser.ReuseRecord = true
114         p := &csvDecoder{
115                 buf:    buf,
116                 types:  types,
117                 header: nil,
118                 reader: parser,
119         }
120         return p
121 }
```

Arquivo csv_encoder.go

```
1  package coder
2
3  import (
4          "encoding/csv"
5          "fmt"
6          "io"
7          "reflect"
8
9          "github.com/pkg/errors"
10 )
11
12 type csvEncoder struct {
13         types      map[reflect.Type]map[int]int
```

```
14          header     []string
15          headerMap  map[string]int
16          writer     *csv.Writer
17          headerSent bool
18  }
19
20  func (cp *csvEncoder) recognize(typ reflect.Type) map[int]int {
21          if mapping, ok := cp.types[typ]; ok {
22                  return mapping
23          }
24          mapping := make(map[int]int)
25          fields := typ.NumField()
26          for c := 0; c < fields; c++ {
27                  field := typ.Field(c)
28                  tag := field.Tag.Get("csv")
29                  if tag == "" {
30                          // Ignore field without a tag
31                          continue
32                  }
33                  headPos, ok := cp.headerMap[tag]
34                  if !ok {
35                          // Ignore field that should not exist in the CSV
36                          continue
37                  }
38                  mapping[headPos] = c
39          }
40          cp.types[typ] = mapping
41          return mapping
42  }
43  func (cp *csvEncoder) Encode(result interface{}) error {
44          var err error
45          if !cp.headerSent {
46                  err = cp.writer.Write(cp.header)
47                  cp.headerSent = true
48          }
49          var value reflect.Value
50          if err == nil {
51                  value, err = validateType(result)
52          }
```

```
53          if err == nil {
54                  row := make([]string, len(cp.header))
55                  resultv := value.Elem()
56                  mapping := cp.recognize(resultv.Type())
57                  for column, field := range mapping {
58                          // Turn the column into a string
59                          row[column] = fmt.Sprint(resultv.Field(field))
60                  }
61                  err = cp.writer.Write(row)
62          }
63          return errors.WithStack(err)
64  }
65
66  func (cp *csvEncoder) Flush() error {
67          cp.writer.Flush()
68          return errors.WithStack(cp.writer.Error())
69  }
70
71  func (cp *csvEncoder) Close() error {
72          return cp.Flush()
73  }
74
75  // NewCSVEncoder returns a Streaming Encoder that is able to encode
    ↪  structs
76  // into CSV format and write it into a stream
77  func NewCSVEncoder(w io.Writer, headers []string) StreamingEncoder {
78          types := make(map[reflect.Type]map[int]int)
79          headerMap := make(map[string]int)
80          for c, header := range headers {
81                  headerMap[header] = c
82          }
83          writer := csv.NewWriter(w)
84          p := &csvEncoder{
85                  types:     types,
86                  header:    headers,
87                  headerMap: headerMap,
88                  writer:    writer,
89          }
90          return p
```

```
91 }
```

Arquivo errors.go

```
1  package coder
2
3  import (
4          "fmt"
5
6          "github.com/pkg/errors"
7  )
8
9  type unexpectedByte struct {
10         b byte
11 }
12
13 func (u unexpectedByte) Error() string {
14         return fmt.Sprintf("%s: '%s'", errUnexpectedByte.Error(),
           ↪  string(u.b))
15 }
16
17 func (u unexpectedByte) Cause() error {
18         return errUnexpectedByte
19 }
20
21 func isErrNotPointer(err error) bool {
22         return err == errNotPointer
23 }
24
25 func isErrNotStruct(err error) bool {
26         return err == errNotStruct
27 }
28
29 func isErrUnexpectedByte(err error) bool {
30         return err == errUnexpectedByte
31 }
32
33 func isErrAlreadyClosed(err error) bool {
34         return err == errAlreadyClosed
35 }
```

```
36
37 // IsCoderError checks if the error is caused by the coder package
38 func IsCoderError(err error) bool {
39         err = errors.Cause(err)
40         return isErrAlreadyClosed(err) || isErrNotPointer(err) ||
           ↪  isErrNotStruct(err) || isErrUnexpectedByte(err)
41 }
42
43 // IsErrNotPointer checks if the error was caused by the value not being
44 // a pointer
45 func IsErrNotPointer(err error) bool {
46         return isErrNotPointer(errors.Cause(err))
47 }
48
49 // IsErrNotStruct checks if the error was caused by the value not being
50 // a struct
51 func IsErrNotStruct(err error) bool {
52         return isErrNotStruct(errors.Cause(err))
53 }
54
55 // IsErrUnexpectedByte checks if the error was caused by a unexpected
   ↪  byte
56 func IsErrUnexpectedByte(err error) bool {
57         return isErrUnexpectedByte(errors.Cause(err))
58 }
59
60 // IsErrAlreadyClosed checks if the error was caused by the streaming
   ↪  encoder
61 // or decoder being already closed
62 func IsErrAlreadyClosed(err error) bool {
63         return isErrAlreadyClosed(errors.Cause(err))
64 }
```

Arquivo gob_coder.go

```
1 package coder
2
3 import (
4         "encoding/gob"
5         "io"
```

```
6           "reflect"
7
8           "github.com/fjorgemota/gurudamatricula/util/pool"
9           "github.com/pkg/errors"
10  )
11
12  /*
13  gobCoder defines a coder that encode and decode values based on Gob
    ↪  Encoding
14  */
15  type gobCoder struct{}
16
17  /*
18  Register registers a new type in GOB Encoder/Decoder
19  */
20  func (cod gobCoder) Register(val interface{}) {
21          gob.Register(val)
22  }
23
24  func (cod gobCoder) DecodeFrom(reader io.Reader, result interface{})
    ↪  error {
25          if reflect.ValueOf(result).Kind() != reflect.Ptr {
26                  return errors.WithStack(errNotPointer)
27          }
28          if e, ok := result.(Decoder); ok {
29                  return e.DecodeFrom(cod, reader)
30          }
31          dec := gob.NewDecoder(reader)
32          err := dec.Decode(result)
33          return err
34
35  }
36
37  /*
38  EncodeTo converts a value to bytes using GOB encoding
39  */
40  func (cod gobCoder) EncodeTo(val interface{}, target io.Writer) error {
41          if e, ok := val.(Encoder); ok {
42                  return e.EncodeTo(cod, target)
```

```
43          }
44          buf := pool.GetBufioWriter(target)
45          defer pool.PutBufioWriter(buf)
46          enc := gob.NewEncoder(buf)
47          err := enc.Encode(val)
48          if err == nil {
49                  err = buf.Flush()
50          }
51          return err
52  }
53
54  /*
55  NewGobCoder returns a new coder which codifies the data using GOB
56  */
57  func NewGobCoder() Coder {
58          return gobCoder{}
59  }
```

Arquivo gob_streaming_decoder.go

```
1   package coder
2
3   import (
4           "bufio"
5           "encoding/gob"
6           "io"
7           "reflect"
8
9           "github.com/fjorgemota/gurudamatricula/util/pool"
10          "github.com/pkg/errors"
11  )
12
13  type gobStreamingDecoder struct {
14          reader  *bufio.Reader
15          decoder *gob.Decoder
16  }
17
18  func (cp *gobStreamingDecoder) decode(result reflect.Value) error {
19          err := cp.decoder.DecodeValue(result)
20          if errors.Cause(err) == io.EOF {
```

```
                pool.PutBufioReader(cp.reader)
                cp.reader = nil
        }
        return err
}

func (cp *gobStreamingDecoder) Decode(result interface{}) error {
        value, err := validateType(result)
        if err == nil {
                err = cp.decode(value)
        }
        return errors.WithStack(err)
}

func (cp *gobStreamingDecoder) Ignore() error {
        return cp.decode(reflect.Value{})
}

// NewGOBStreamingDecoder returns a Streaming Decoder that is able to
↪  decode GOB format
// into structs
func NewGOBStreamingDecoder(r io.Reader) StreamingDecoder {
        reader := pool.GetBufioReader(r)
        p := &gobStreamingDecoder{
                reader:  reader,
                decoder: gob.NewDecoder(reader),
        }
        return p
}
```

Arquivo gob_streaming_encoder.go

```
package coder

import (
        "bufio"
        "encoding/gob"
        "io"
        "reflect"

```

```
9           "github.com/fjorgemota/gurudamatricula/util/pool"
10  )
11
12  /*
13  *gobStreamingEncoder defines a coder that encode and decode values based
    ↪  on Gob Encoding
14   */
15  type gobStreamingEncoder struct {
16          encoder    *gob.Encoder
17          buf        *bufio.Writer
18          origWriter io.WriteCloser
19  }
20
21  func (cod *gobStreamingEncoder) Encode(value interface{}) error {
22          var err error
23          if cod.buf == nil {
24                  err = errAlreadyClosed
25          }
26          var rvalue reflect.Value
27          if err == nil {
28                  rvalue, err = validateType(value)
29          }
30          if err == nil {
31                  err = cod.encoder.EncodeValue(rvalue)
32          }
33          return err
34  }
35
36  func (cod *gobStreamingEncoder) Flush() error {
37          var err error
38          if cod.buf == nil {
39                  err = errAlreadyClosed
40          }
41          if err == nil {
42                  err = cod.buf.Flush()
43          }
44          return err
45  }
46
```

```
47 func (cod *gobStreamingEncoder) Close() error {
48         err := cod.Flush()
49         if err == nil {
50                 err = cod.origWriter.Close()
51         }
52         if err == nil {
53                 pool.PutBufioWriter(cod.buf)
54                 cod.buf = nil
55         }
56         return err
57 }
58
59 /*
60 NewGOBStreamingEncoder returns a new coder which codifies the data using
   ↪  GOB
61 */
62 func NewGOBStreamingEncoder(writer io.WriteCloser) StreamingEncoder {
63         buf := pool.GetBufioWriter(writer)
64         return &gobStreamingEncoder{
65                 encoder:    gob.NewEncoder(buf),
66                 buf:        buf,
67                 origWriter: writer,
68         }
69 }
```

Arquivo hashid_encoder.go

```
1  package coder
2
3  import "github.com/speps/go-hashids"
4
5  type hashidEncoder struct {
6          hasher *hashids.HashID
7  }
8
9  func (he hashidEncoder) Encode(n int) (string, error) {
10         return he.EncodeInt64(int64(n))
11 }
12
13 func (he hashidEncoder) EncodeInt64(n int64) (string, error) {
```

```
14          var tmp [1]int64
15          tmp[0] = n
16          return he.hasher.EncodeInt64(tmp[:])
17  }
18
19  func NewHashIDEncoder(hasher *hashids.HashID) IntEncoder {
20          return hashidEncoder{hasher: hasher}
21  }
```

Arquivo json_coder.go

```
1   package coder
2
3   import (
4           "encoding/json"
5           "io"
6           "reflect"
7
8           "github.com/pkg/errors"
9   )
10
11  /*
12  jsonCoder defines a coder that encode and decode values based on Gob
    ↪  Encoding
13  */
14  type jsonCoder struct{}
15
16  /*
17  Register registers a new type in GOB Encoder/Decoder
18  */
19  func (cod jsonCoder) Register(val interface{}) {}
20
21  func (cod jsonCoder) DecodeFrom(reader io.Reader, result interface{})
    ↪  error {
22          if reflect.ValueOf(result).Kind() != reflect.Ptr {
23                  return errors.WithStack(errNotPointer)
24          }
25          if d, ok := result.(Decoder); ok {
26                  return d.DecodeFrom(cod, reader)
27          }
```

```
28          dec := json.NewDecoder(reader)
29          return dec.Decode(result)
30  }
31
32  func (cod jsonCoder) EncodeTo(result interface{}, target io.Writer) error
    ↪ {
33          if d, ok := result.(Encoder); ok {
34                  return d.EncodeTo(cod, target)
35          }
36          enc := json.NewEncoder(target)
37          return enc.Encode(result)
38  }
39
40  /*
41  NewJSONCoder returns a new coder which codifies the data using GOB
42  */
43  func NewJSONCoder() Coder {
44          return jsonCoder{}
45  }
```

Arquivo json_decoder.go

```
1   package coder
2
3   import (
4           "bufio"
5           "io"
6
7           "github.com/fjorgemota/gurudamatricula/util/pool"
8           "github.com/pkg/errors"
9           "github.com/ugorji/go/codec"
10  )
11
12  type jsonDecoder struct {
13          started bool
14          ended   bool
15          reader  *bufio.Reader
16          decoder *codec.Decoder
17  }
18
```

```
19 func (cp *jsonDecoder) decode(result interface{}) error {
20         var b byte
21         var state int
22         var err error
23         // 0: starting
24         // 1: validate document (starting)
25         // 2: validate document (continuing)
26         // 3: finishing
27         // 4: Validated!
28         for err == nil && state < 3 && !cp.ended {
29                 b, err = cp.reader.ReadByte()
30                 if (b == ' ' || b == '\n') && err == nil {
31                         continue
32                 }
33                 if cp.started {
34                         if b == ']' && (state == 0 || state == 1) {
35                                 state = 3
36                         } else if b == ',' && state == 0 {
37                                 state = 2
38                         } else if (state == 1 || state == 2) && b == '{'
                           ↪ {
39                                 state = 4
40                         } else if err == nil {
41                                 err = unexpectedByte{b}
42                         }
43                 } else {
44                         if b == '[' && state == 0 {
45                                 cp.started = true
46                                 state = 1
47                         } else if err == nil {
48                                 err = cp.reader.UnreadByte()
49                                 if err == nil {
50                                         err = unexpectedByte{b}
51                                 }
52                         }
53                 }
54         }
55         if err == nil {
56                 switch state {
```

```
                case 3:
                        err = io.EOF
                case 4:
                        err = cp.reader.UnreadByte()
                }
        }
        if err == nil {
                err = cp.decoder.Decode(result)
        }
        if err == io.EOF {
                cp.ended = true
        }
        if cp.ended {
                pool.PutBufioReader(cp.reader)
                cp.reader = nil
                cp.decoder = nil
                err = io.EOF
        }
        return errors.WithStack(err)
}

func (cp *jsonDecoder) Decode(result interface{}) error {
        _, err := validateType(result)
        if err == nil {
                err = cp.decode(result)
        }
        return err
}

func (cp *jsonDecoder) Ignore() error {
        var data interface{}
        err := cp.decode(&data)
        return err
}

// NewJSONDecoder returns a Streaming Decoder that is able to decode JSON
↪  format
// into structs
func NewJSONDecoder(r io.Reader) StreamingDecoder {
```

```
95          handler := &codec.JsonHandle{}
96          handler.InternString = true
97          reader := pool.GetBufioReader(r)
98          decoder := codec.NewDecoder(reader, handler)
99          p := &jsonDecoder{
100                 reader:  reader,
101                 decoder: decoder,
102         }
103         return p
104 }
```

Arquivo json_encoder.go

```
1  package coder
2
3  import (
4          "bufio"
5          "io"
6
7          "github.com/fjorgemota/gurudamatricula/util/pool"
8          "github.com/pkg/errors"
9          "github.com/ugorji/go/codec"
10 )
11
12 type jsonEncoder struct {
13         started    bool
14         ended      bool
15         origWriter io.WriteCloser
16         writer     *bufio.Writer
17         enc        *codec.Encoder
18 }
19
20 func (cp *jsonEncoder) Encode(result interface{}) error {
21         var err error
22         if cp.ended {
23                 err = errAlreadyClosed
24         }
25         if err == nil {
26                 if cp.started {
27                         err = cp.writer.WriteByte(',')
```

```
                } else {
                        err = cp.writer.WriteByte('[')
                        cp.started = true
                }
        }
        if err == nil {
                _, err = validateType(result)
        }
        if err == nil {
                err = cp.enc.Encode(result)
        }
        return errors.WithStack(err)
}

func (cp *jsonEncoder) Flush() error {
        var err error
        if cp.writer == nil || cp.ended {
                err = cp.writer.Flush()
        }
        return errors.WithStack(err)
}

func (cp *jsonEncoder) Close() error {
        var err error
        if cp.ended {
                err = errAlreadyClosed
        }
        if !cp.started && err == nil {
                err = cp.writer.WriteByte('[')
        }
        if err == nil {
                err = cp.writer.WriteByte(']')
                cp.ended = true
        }
        if err == nil {
                err = cp.Flush()
        }
        if err == nil {
                err = cp.origWriter.Close()
```

```
67          }
68          if err == nil {
69                  pool.PutBufioWriter(cp.writer)
70                  cp.writer = nil
71          }
72          return errors.WithStack(err)
73  }
74
75  // NewJSONEncoder returns a Streaming Encoder that is able to encode
    ↪  structs
76  // into JSON format and write it into a stream
77  func NewJSONEncoder(w io.WriteCloser) StreamingEncoder {
78          var handler codec.Handle = new(codec.JsonHandle)
79          writer := pool.GetBufioWriter(w)
80          p := &jsonEncoder{
81                  origWriter: w,
82                  writer:     writer,
83                  enc:        codec.NewEncoder(writer, handler),
84          }
85          return p
86  }
```

Arquivo json_native_decoder.go

```
1   package coder
2
3   import (
4           "bufio"
5           "encoding/json"
6           "io"
7           "reflect"
8
9           "github.com/fjorgemota/gurudamatricula/util/pool"
10          "github.com/pkg/errors"
11  )
12
13  type jsonNativeDecoder struct {
14          started bool
15          ended   bool
16          decoder *json.Decoder
```

```
17          reader  *bufio.Reader
18  }
19
20  func (cp *jsonNativeDecoder) decode(result interface{}) error {
21          var err error
22          for !cp.ended && err == nil {
23                  if !cp.started {
24                          var token json.Token
25                          token, err = cp.decoder.Token()
26                          if delim, ok := token.(json.Delim); ok {
27                                  if delim != '[' {
28                                          err = unexpectedByte{byte(delim)}
29                                  } else {
30                                          cp.started = true
31                                  }
32                          }
33                  } else if !cp.decoder.More() {
34                          cp.ended = true
35                          pool.PutBufioReader(cp.reader)
36                          cp.decoder = nil
37                          err = io.EOF
38                  } else {
39                          break
40                  }
41          }
42          if !cp.ended && err == nil {
43                  err = cp.decoder.Decode(result)
44          }
45          return errors.WithStack(err)
46  }
47
48  func (cp *jsonNativeDecoder) Decode(result interface{}) error {
49          var err error
50          reflection := reflect.ValueOf(result)
51          if reflection.Kind() != reflect.Ptr {
52                  err = errNotPointer
53          }
54          if err == nil {
55                  err = cp.decode(result)
```

```
        }
        return err
}

func (cp *jsonNativeDecoder) Ignore() error {
        var data interface{}
        err := cp.decode(&data)
        return err
}

// NewJSONNativeDecoder returns a Streaming Decoder that is able to
↪  decode JSON format
// into structs
func NewJSONNativeDecoder(r io.Reader) StreamingDecoder {
        reader := pool.GetBufioReader(r)
        decoder := json.NewDecoder(reader)
        // decoder.UseNumber()
        p := &jsonNativeDecoder{
                decoder: decoder,
                reader:  reader,
        }
        return p
}
```

Arquivo splitter_encoder.go

```
package coder

type splitterEncoder struct {
        create  func() (StreamingEncoder, error)
        actual  StreamingEncoder
        counter int
        limit   int
}

func (se *splitterEncoder) Encode(val interface{}) error {
        var err error
        if se.limit > 0 && se.limit <= se.counter {
                err = se.Close()
        }
```

```
15          if err == nil && se.actual == nil {
16                  se.actual, err = se.create()
17          }
18          if err == nil {
19                  err = se.actual.Encode(val)
20          }
21          if err == nil {
22                  se.counter++
23          }
24          return err
25  }
26
27  func (se *splitterEncoder) Flush() error {
28          var err error
29          if se.actual != nil {
30                  err = se.actual.Flush()
31          }
32          return err
33  }
34
35  func (se *splitterEncoder) Close() error {
36          err := se.Flush()
37          if se.actual != nil {
38                  err = se.actual.Close()
39                  if err == nil {
40                          se.counter = 0
41                          se.actual = nil
42                  }
43          }
44          return err
45  }
46
47  func NewSplitterEncoder(create func() (StreamingEncoder, error), limit
    ↪  int) StreamingEncoder {
48          return &splitterEncoder{
49                  create: create,
50                  limit:  limit,
51          }
52  }
```

Arquivo validator.go

```
package coder

import "reflect"

func validateType(value interface{}) (reflect.Value, error) {
        var err error
        reflection := reflect.ValueOf(value)
        if reflection.Kind() != reflect.Ptr {
                err = errNotPointer
        }
        if err == nil && reflection.Elem().Kind() != reflect.Struct {
                err = errNotStruct
        }
        return reflection, err
}
```

Arquivo zlib.go

```
package coder

import (
        "compress/zlib"
        "io"
)

type zlibHandle struct {
        level int
}

func (zh zlibHandle) Writer(w io.Writer) (io.WriteCloser, error) {
        return zlib.NewWriterLevel(w, zh.level)
}
func (zh zlibHandle) Reader(r io.Reader) (io.ReadCloser, error) {
        return zlib.NewReader(r)
}

func NewZLIBHandle(level int) CompressorHandle {
```

```
20          return zlibHandle{level: level}
21  }
```

B.1.24 Pasta util/dispatcher

Arquivo base.go

```
1   package dispatcher
2
3   import (
4           "io"
5           "reflect"
6   )
7
8   var errorType = reflect.TypeOf((*error)(nil)).Elem()
9
10  // Dispatcher represents a generic dispatcher that can encode calls to
    ↪  functions
11  // and decode these calls calling the respective function automatically
12  type Dispatcher interface {
13          GetType(name string) (reflect.Type, error)
14          Register(name string, fn interface{}) error
15          Decode(reader io.Reader, args ...interface{}) (bool, error)
16          Encode(writer io.Writer, name string, args ...interface{}) error
17  }
18
19  // Handler represents an validator and processor for the calls made by
    ↪  the
20  // dispatcher, so the user can customize what types of functions are
    ↪  accepted
21  // and how to manage arguments easily
22  type Handler interface {
23          ValidateFunction(fn reflect.Type) error
24          ProcessCall(fn reflect.Type, args []interface{}) ([]interface{},
            ↪  error)
25          ProcessArgs(fn reflect.Type, oldArgs, newArgs []interface{})
            ↪  ([]interface{}, error)
26          ProcessResult(fn reflect.Type, args, newArgs []interface{},
            ↪  result []reflect.Value, panicValue interface{}) error
27  }
```

```
28
29 type invocation struct {
30         Name string
31         Args []interface{}
32 }
```

Arquivo dispatcher.go

```
1  package dispatcher
2
3  import (
4          "errors"
5          "fmt"
6          "io"
7          "reflect"
8          "runtime"
9
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11 )
12
13 type dispatcher struct {
14         functions map[string]reflect.Value
15         keyToFile map[string]string
16         cod       coder.Coder
17         handler   Handler
18 }
19
20 func (fd *dispatcher) GetType(name string) (reflect.Type, error) {
21         value, err := fd.getTask(name)
22         var typ reflect.Type
23         if err == nil {
24                 typ = value.Type()
25         }
26         return typ, err
27 }
28
29 func (fd *dispatcher) Register(name string, fn interface{}) error {
30         rv := reflect.ValueOf(fn)
31         rt := rv.Type()
32         if rt.Kind() != reflect.Func {
```

```
33                  return newTaskError(name, errors.New("fn is not a
                    ↪  function, but it should be"))
34          }
35          err := fd.handler.ValidateFunction(rt)
36          _, file, _, _ := runtime.Caller(1)
37          if old, ok := fd.keyToFile[name]; ok {
38                  err = newTaskError(name, fmt.Errorf("task defined in two
                    ↪  files: %s and %s", old, file))
39          }
40          for i := 0; i < rt.NumIn(); i++ {
41                  // Only concrete types may be registered. If the argument
                    ↪  has
42                  // interface type, the client is resposible for
                    ↪  registering the
43                  // concrete types it will hold.
44                  if err == nil || rt.In(i).Kind() == reflect.Interface {
45                          continue
46                  }
47                  fd.cod.Register(reflect.Zero(rt.In(i)).Interface())
48          }
49          if err == nil {
50                  fd.keyToFile[name] = file
51                  fd.functions[name] = rv
52          }
53          return err
54  }
55
56  func (fd *dispatcher) getTask(name string) (reflect.Value, error) {
57          f, ok := fd.functions[name]
58          if !ok {
59                  return f, newTaskError(name, errors.New("task does not
                    ↪  exist"))
60          }
61          return f, nil
62  }
63
64  func (fd *dispatcher) Encode(writer io.Writer, name string, args
    ↪  ...interface{}) error {
65          f, err := fd.getTask(name)
```

```
66          if err == nil {
67                  args, err = fd.handler.ProcessCall(f.Type(), args)
68          }
69          if err == nil {
70                  inv := invocation{
71                          Name: name,
72                          Args: args,
73                  }
74                  err = fd.cod.EncodeTo(inv, writer)
75          }
76          return newTaskError(name, err)
77  }
78
79  func (fd *dispatcher) getZeroFromArgument(f reflect.Type, i int)
    ↪  reflect.Value {
80          // Task was passed a nil argument, so we must construct
81          // the zero value for the argument here.
82          var at reflect.Type
83          if !f.IsVariadic() || i < f.NumIn()-1 {
84                  at = f.In(i)
85          } else {
86                  at = f.In(f.NumIn() - 1).Elem()
87          }
88          return reflect.Zero(at)
89  }
90  func (fd *dispatcher) Decode(reader io.Reader, args ...interface{})
    ↪  (shouldRetry bool, err error) {
91          var inv invocation
92          err = fd.cod.DecodeFrom(reader, &inv)
93          var f reflect.Value
94          if err == nil {
95                  f, err = fd.getTask(inv.Name)
96          }
97          if err == nil {
98                  inv.Args, err = fd.handler.ProcessArgs(f.Type(), inv.Args,
                    ↪  args)
99          }
100         if err == nil {
101                 ft := f.Type()
```

```
102                 in := make([]reflect.Value, len(inv.Args))
103                 for n, arg := range inv.Args {
104                         var v reflect.Value
105                         if arg != nil {
106                                 v = reflect.ValueOf(arg)
107                         } else {
108                                 v = fd.getZeroFromArgument(ft, n)
109                         }
110                         in[n] = v
111                 }
112                 shouldRetry = true
113                 defer func() {
114                         if panicValue := recover(); panicValue != nil {
115                                 err = fd.handler.ProcessResult(ft,
                                    ↪  inv.Args, args, nil, panicValue)
116                                 err = newTaskError(inv.Name, err)
117                         }
118                 }()
119                 out := f.Call(in)
120                 err = fd.handler.ProcessResult(ft, inv.Args, args, out,
                    ↪  nil)
121                 if n := ft.NumOut(); n > 0 && ft.Out(n-1) == errorType {
122                         if errv := out[n-1]; !errv.IsNil() && err == nil
                            ↪  {
123                                 err = errv.Interface().(error)
124                         }
125                 }
126                 if err == nil {
127                         shouldRetry = false
128                 }
129         }
130         return shouldRetry, newTaskError(inv.Name, err)
131 }
132
133 // NewDispatcher returns a new task dispatcher
134 func NewDispatcher(cod coder.Coder, handler Handler) Dispatcher {
135         return &dispatcher{
136                 cod:       cod,
137                 functions: make(map[string]reflect.Value),
```

```
138                 keyToFile: make(map[string]string),
139                 handler:   handler,
140         }
141 }
```

Arquivo errors.go

```
1  package dispatcher
2
3  import (
4          "fmt"
5
6          "github.com/pkg/errors"
7  )
8
9  type taskError struct {
10         task string
11         err  error
12 }
13
14 func (t taskError) Error() string {
15         return fmt.Sprintf("dispatcher: error on task '%s': %s", t.task,
           ↪  t.err.Error())
16 }
17
18 func (t taskError) Cause() error {
19         return t.err
20 }
21
22 func newTaskError(task string, err error) error {
23         if err == nil {
24                 return nil
25         }
26         return errors.WithStack(taskError{task: task, err: err})
27 }
28
29 // IsTaskError checks if the error was caused by a task
30 func IsTaskError(err error) bool {
31         type cause interface {
32                 Cause() error
```

```
33          }
34  startCheck:
35          _, isTask := err.(taskError)
36          if isTask {
37                  return true
38          }
39          c, ok := err.(cause)
40          if ok {
41                  err = c.Cause()
42                  goto startCheck
43          }
44          return ok
45  }
```

Arquivo handler.go

```
1   package dispatcher
2
3   import (
4           "fmt"
5           "reflect"
6   )
7
8   // BasicHandler implements a basic Handler that validates calls to
    ↪  functions in
9   // a Dispatcher by only expecting respect for the number of the arguments
    ↪  and
10  // it's types
11  type BasicHandler struct{}
12
13  // ValidateFunction always returns nil
14  func (h BasicHandler) ValidateFunction(ft reflect.Type) error {
15          return nil
16  }
17
18  // ProcessCall validates if a function is valid by checking the number
    ↪  and
19  // the types of the arguments
20  func (h BasicHandler) ProcessCall(ft reflect.Type, args []interface{})
    ↪  ([]interface{}, error) {
```

```
21          err := ValidateMinArguments(ft, len(args))
22          if err == nil {
23                  err = ValidateMaxArguments(ft, len(args))
24          }
25          if err == nil {
26                  err = ValidateCompatibleArgumentTypes(ft, args)
27          }
28          return args, err
29  }
30
31  // ProcessArgs just return the old arguments slice to the user
32  func (h BasicHandler) ProcessArgs(ft reflect.Type, oldArgs []interface{},
    ↪  newArgs []interface{}) ([]interface{}, error) {
33          return oldArgs, nil
34  }
35
36  // ProcessResult is a no-op that just returns nil
37  func (h BasicHandler) ProcessResult(ft reflect.Type, args, newArgs
    ↪  []interface{}, result []reflect.Value, panicValue interface{}) error
    ↪  {
38          var err error
39          if panicValue != nil {
40                  var ok bool
41                  if err, ok = panicValue.(error); !ok {
42                          err = fmt.Errorf("panicked with %s", panicValue)
43                  }
44          }
45          return err
46  }
47
48  var _ Handler = BasicHandler{}
```

Arquivo types.go

```
1   package dispatcher
2
3   import (
4           "errors"
5           "fmt"
6           "reflect"
```

```
)

// HasMinArguments checks if the function passed as reflect.Type has the
↪  minimum
// of minArgs arguments needed for the execution of the function
func HasMinArguments(fn reflect.Type, minArgs int) (bool, error) {
        if fn.Kind() != reflect.Func {
                return false, errors.New("fn is not a type of a function,
                ↪  but it should be")
        }
        args := fn.NumIn()
        if fn.IsVariadic() {
                args--
        }
        return minArgs >= args, nil
}

// HasMaxArguments checks if the function passed as reflect.Type has the
↪  maximum
// of maxArgs arguments needed for the execution of the function
func HasMaxArguments(fn reflect.Type, maxArgs int) (bool, error) {
        if fn.Kind() != reflect.Func {
                return false, errors.New("fn is not a type of a function,
                ↪  but it should be")
        }
        if fn.IsVariadic() {
                // Cannot really confirm that
                return true, nil
        }
        return maxArgs <= fn.NumIn(), nil
}

// GetArgumentTypes returns a slice containing the types of the arguments
↪  that
// the function type fn has
func GetArgumentTypes(fn reflect.Type) ([]reflect.Type, error) {
        if fn.Kind() != reflect.Func {
                return nil, errors.New("fn is not a type of a function,
                ↪  but it should be")
```

```
        }
        fnArgs := make([]reflect.Type, fn.NumIn())
        for i := range fnArgs {
                fnArgs[i] = fn.In(i)
        }
        return fnArgs, nil
}

// AdaptVariadic returns a slice with the same length of the slice args
// based on the behavior of a variadic function in Go
func AdaptVariadic(fnArgs []reflect.Type, nArgs int) []reflect.Type {
        fnArgsLen := len(fnArgs) - 1
        result := make([]reflect.Type, nArgs)
        for i := 0; i < nArgs; i++ {
                if fnArgsLen <= i {
                        // Variadic arg
                        result[i] = fnArgs[fnArgsLen].Elem()
                } else {
                        // Non variadic arg
                        result[i] = fnArgs[i]
                }
        }
        return result
}

// HasCompatibleTypes returns false and an error if at least one variable
↪  in
// slice args is not compatible with the respective type on slice fnArgs
func HasCompatibleTypes(fnArgs []reflect.Type, args []interface{}) (bool,
↪  error) {
        if len(fnArgs) != len(args) {
                return false, errors.New("arguments should have the same
                ↪  size")
        }
        for i := range args {
                at := reflect.TypeOf(args[i])
                dt := fnArgs[i]
                // nil arguments won't have a type, so they need special
                ↪  handling.
```

```
75                  if at == nil {
76                          // nil interface
77                          switch dt.Kind() {
78                          case reflect.Chan, reflect.Func,
                            ↪  reflect.Interface, reflect.Map, reflect.Ptr,
                            ↪  reflect.Slice:
79                                  continue // may be nil
80                          }
81                          return false, fmt.Errorf("argument %d has wrong
                            ↪  type: %v is not nilable", i, dt)
82                  }
83                  switch at.Kind() {
84                  case reflect.Chan, reflect.Func, reflect.Interface,
                    ↪  reflect.Map, reflect.Ptr, reflect.Slice:
85                          av := reflect.ValueOf(args[i])
86                          if av.IsNil() {
87                                  args[i] = nil
88                          }
89                  }
90                  if !at.AssignableTo(dt) {
91                          return false, fmt.Errorf("argument %d has wrong
                            ↪  type: %v is not assignable to %v", i, at, dt)
92                  }
93          }
94          return true, nil
95  }
96
97  // HasCompatibleArgumentTypes checks if a function type fn has arguments
98  // compatible the respective value in args slice
99  func HasCompatibleArgumentTypes(fn reflect.Type, args []interface{})
    ↪  (bool, error) {
100         if fn.Kind() != reflect.Func {
101                 return false, errors.New("fn is not a type of a function,
                    ↪  but it should be")
102         }
103         var result bool
104         fnArgs, err := GetArgumentTypes(fn)
105         if err == nil && fn.IsVariadic() && len(args) >= len(fnArgs)-1 {
106                 fnArgs = AdaptVariadic(fnArgs, len(args))
```

```
107         }
108         if err == nil {
109                 result, err = HasCompatibleTypes(fnArgs, args)
110         }
111         return result, err
112 }
```

Arquivo validation.go

```
1  package dispatcher
2
3  import (
4          "fmt"
5          "reflect"
6  )
7
8  // ValidateMinArguments returns an error if the function type fn do not
   ↪  have
9  // at least minArgs arguments
10 func ValidateMinArguments(fn reflect.Type, minArgs int) error {
11         ok, err := HasMinArguments(fn, minArgs)
12         if !ok && err == nil {
13                 err = fmt.Errorf("too few arguments to func: %d > %d",
                   ↪  fn.NumIn(), minArgs)
14         }
15         return err
16 }
17
18 // ValidateMaxArguments returns an error if the function type fn do not
   ↪  have
19 // at maximum minArgs arguments
20 func ValidateMaxArguments(fn reflect.Type, maxArgs int) error {
21         ok, err := HasMaxArguments(fn, maxArgs)
22         if !ok && err == nil {
23                 err = fmt.Errorf("too many arguments to func: %d < %d",
                   ↪  fn.NumIn(), maxArgs)
24         }
25         return err
26 }
27
```

```
28 // ValidateCompatibleTypes returns an error if the at least one type on
29 // slice fnArgs is not compatible with the respective value on args slice
30 func ValidateCompatibleTypes(fnArgs []reflect.Type, args []interface{})
   ↪  error {
31         _, err := HasCompatibleTypes(fnArgs, args)
32         return err
33 }
34
35 // ValidateCompatibleArgumentTypes returns an error if the at least one
   ↪  type of
36 // the argument of the function type fn is not compatible with the
   ↪  respective
37 // value on args slice
38 func ValidateCompatibleArgumentTypes(fn reflect.Type, args []interface{})
   ↪  error {
39         _, err := HasCompatibleArgumentTypes(fn, args)
40         return err
41 }
```

B.1.25 Pasta util/file

Arquivo base.go

```
1  package file
2
3  import (
4          "errors"
5          "io"
6  )
7
8  var (
9          // ErrNotExist is returned when the path does not exist
10         errNotExist = errors.New("file: Path does not exists")
11         // ErrEmpty is returned when the file informed is a empty string
12         errEmpty = errors.New("file: You cannot inform a empty file
           ↪  path")
13         // ErrDone is returned when there are no more results to return
14         errDone = errors.New("file: Query has no more results")
15 )
16
```

```
17 // ReaderSeekerCloser is a interface that have methods to read, seek and
   ↪  close
18 // readers..
19 type ReaderSeekerCloser interface {
20         io.ReadSeeker
21         io.Closer
22 }
23
24 // Manager is a interface of a struct that manages a file on a
25 // filesystem
26 type Manager interface {
27         Reader(file string) (ReaderSeekerCloser, error)
28         Writer(file string) (io.WriteCloser, error)
29         Delete(file string) error
30         Iterator(prefix string) (Iterator, error)
31 }
32
33 // Iterator allows the user to iterate by all files managed by a Manager
34 type Iterator interface {
35         Next() (string, error)
36 }
```

Arquivo deleter.go

```
1  package file
2
3  // DeleteFiles deletes all the files specified automatically
4  func DeleteFiles(manager Manager, filenames []string) error {
5          var err error
6          for _, filename := range filenames {
7                  if err == nil {
8                          err = manager.Delete(filename)
9                  }
10         }
11         return err
12 }
```

Arquivo errors.go

```
1  package file
2
3  import (
4          "fmt"
5
6          "github.com/pkg/errors"
7  )
8
9  type fileError struct {
10         file string
11         err  error
12 }
13
14 func (f fileError) Error() string {
15         return fmt.Sprintf("file: error on file '%s': %s", f.file,
           ↪  f.err.Error())
16 }
17
18 func (f fileError) Cause() error {
19         return f.err
20 }
21
22 func newFileError(file string, err error) error {
23         if err == nil {
24                 return err
25         }
26         return errors.WithStack(fileError{file: file, err: err})
27 }
28
29 func IsDoneError(err error) bool {
30         return errors.Cause(err) == errDone
31 }
32
33 func IsNotExistError(err error) bool {
34         return errors.Cause(err) == errNotExist
35 }
```

Arquivo gcs.go

```
1  package file
```

```
2
3  import (
4          "io"
5          "os"
6          "path"
7          "sync"
8
9          "google.golang.org/api/iterator"
10
11         "cloud.google.com/go/storage"
12         "golang.org/x/net/context"
13 )
14
15 type gcsIterator struct {
16         it      *storage.ObjectIterator
17         manager gcsManager
18         c       int
19 }
20
21 func (i *gcsIterator) formatName(name string) string {
22         l := len(i.manager.base)
23         name = name[l:]
24         if name[0] == '/' {
25                 name = name[1:]
26         }
27         return name
28 }
29
30 func (i *gcsIterator) Next() (string, error) {
31         next, err := i.it.Next()
32         if err == iterator.Done {
33                 return "", errDone
34         }
35         var actual string
36         if err == nil {
37                 actual = i.formatName(next.Name)
38                 i.c++
39         }
40         return actual, err
```

```
41 }
42
43 type gcsReader struct {
44         obj          *storage.ObjectHandle
45         ctx          context.Context
46         reader, meta *storage.Reader
47         position     int64
48         lock         sync.Mutex
49 }
50
51 func (r *gcsReader) Close() error {
52         r.lock.Lock()
53         defer r.lock.Unlock()
54         if r.reader == nil {
55                 return nil
56         }
57         err := r.reader.Close()
58         r.reader = nil
59         return err
60 }
61
62 func (r *gcsReader) Read(p []byte) (int, error) {
63         r.lock.Lock()
64         defer r.lock.Unlock()
65         var err error
66         if r.reader == nil {
67                 r.reader, err = r.obj.NewRangeReader(r.ctx, r.position,
                  ↪ -1)
68         }
69         var n int
70         if err == nil {
71                 n, err = r.reader.Read(p)
72         }
73         r.position += int64(n)
74         return n, err
75 }
76
77 func (r *gcsReader) Seek(pos int64, whence int) (int64, error) {
78         r.lock.Lock()
```

```
79          defer r.lock.Unlock()
80          oldPos := r.position
81          newPos := r.position
82          switch whence {
83          case os.SEEK_SET:
84                  newPos = pos
85          case os.SEEK_CUR:
86                  newPos += pos
87          case os.SEEK_END:
88                  reader := r.reader
89                  if reader == nil {
90                          reader = r.meta
91                  }
92                  newPos = reader.Size() + pos
93          }
94          if newPos < 0 {
95                  return r.position, os.ErrInvalid
96          }
97          var err error
98          if oldPos != newPos {
99                  // Resets reader..
100                 if r.reader != nil {
101                         // Close if something is available..
102                         err = r.reader.Close()
103                 }
104                 r.reader = nil
105                 r.position = newPos
106         }
107         return r.position, err
108 }
109
110 type gcsManager struct {
111         base   string
112         bucket *storage.BucketHandle
113         ctx    context.Context
114 }
115
116 func (m gcsManager) Reader(file string) (ReaderSeekerCloser, error) {
117         if file == "" {
```

```
118                 return nil, errEmpty
119         }
120         handler := m.bucket.Object(path.Join(m.base, file))
121         // Check if the file exists using a HEAD request..
122         r, err := handler.NewRangeReader(m.ctx, 0, 0)
123         var reader ReaderSeekerCloser
124         reader = &gcsReader{
125                 ctx:  m.ctx,
126                 obj:  handler,
127                 meta: r,
128         }
129         if err == storage.ErrObjectNotExist {
130                 err = errNotExist
131                 reader = nil
132         }
133         return reader, newFileError(file, err)
134 }
135
136 func (m gcsManager) Writer(file string) (io.WriteCloser, error) {
137         if file == "" {
138                 return nil, newFileError(file, errEmpty)
139         }
140         handler := m.bucket.Object(path.Join(m.base, file))
141         return handler.NewWriter(m.ctx), nil
142 }
143
144 func (m gcsManager) Delete(file string) error {
145         if file == "" {
146                 return newFileError(file, errEmpty)
147         }
148         handler := m.bucket.Object(path.Join(m.base, file))
149         err := handler.Delete(m.ctx)
150         if err == storage.ErrObjectNotExist {
151                 err = errNotExist
152         }
153         return newFileError(file, err)
154 }
155
156 func (m gcsManager) Iterator(prefix string) (Iterator, error) {
```

```
157         it := m.bucket.Objects(
158                 m.ctx,
159                 &storage.Query{
160                         Prefix: path.Join(m.base, prefix),
161                 },
162         )
163         iterator := &gcsIterator{
164                 it:      it,
165                 manager: m,
166         }
167         return iterator, nil
168 }
169
170 // NewGCSManager returns a Manager that manages files in GCS
171 func NewGCSManager(base string, ctx context.Context, bucket
    ↪  *storage.BucketHandle) Manager {
172         return gcsManager{
173                 base:   base,
174                 ctx:    ctx,
175                 bucket: bucket,
176         }
177 }
```

Arquivo local.go

```
1  package file
2
3  import (
4          "io"
5          "io/ioutil"
6          "os"
7          "path"
8          "strings"
9          "sync"
10 )
11
12 type localManager struct {
13         base string
14 }
15
```

```
type localIterator struct {
        files  []os.FileInfo
        lock   sync.Mutex
        c      int
        prefix string
}

func (i *localIterator) Next() (string, error) {
        i.lock.Lock()
        defer i.lock.Unlock()
        var actual string
        for !strings.HasPrefix(actual, i.prefix) || actual == "" {
                if len(i.files) <= i.c {
                        return "", errDone
                }
                actual = i.files[i.c].Name()
                i.c++
        }
        return actual, nil
}

func (m localManager) Reader(file string) (ReaderSeekerCloser, error) {
        if file == "" {
                return nil, errEmpty
        }
        reader, err := os.Open(path.Join(m.base, file))
        if os.IsNotExist(err) {
                err = errNotExist
        }
        return reader, newFileError(file, err)
}

func (m localManager) Writer(file string) (io.WriteCloser, error) {
        if file == "" {
                return nil, newFileError(file, errEmpty)
        }
        writer, err := os.Create(path.Join(m.base, file))
        return writer, newFileError(file, err)
}
```

```
55
56 func (m localManager) Delete(file string) error {
57         if file == "" {
58                 return newFileError(file, errEmpty)
59         }
60         err := os.Remove(path.Join(m.base, file))
61         if os.IsNotExist(err) {
62                 err = errNotExist
63         }
64         return newFileError(file, err)
65 }
66
67 func (m localManager) Iterator(prefix string) (Iterator, error) {
68         files, err := ioutil.ReadDir(m.base)
69         var instance Iterator
70         if err == nil {
71                 instance = &localIterator{files: files, prefix: prefix}
72         }
73         return instance, err
74 }
75
76 // NewLocalManager returns a manager that returns objects
77 // managing the local filesystem
78 func NewLocalManager(base string) (Manager, error) {
79         st, err := os.Stat(base)
80         if os.IsNotExist(err) || !st.IsDir() {
81                 return nil, errNotExist
82         }
83         return localManager{base}, nil
84 }
```

Arquivo memory.go

```
1 package file
2
3 import (
4         "bytes"
5         "io"
6         "strings"
7         "sync"
```

```
8
9           "github.com/fjorgemota/gurudamatricula/util/pool"
10  )
11
12  type bufCloser struct {
13          writer *bytes.Buffer
14          reader *bytes.Reader
15          parent *memoryManager
16          name   string
17  }
18
19  func (buf *bufCloser) Close() error {
20          if buf.writer != nil {
21                  buf.parent.lock.Lock()
22                  defer buf.parent.lock.Unlock()
23                  buf.parent.files[buf.name] = make([]byte,
                    ↪  buf.writer.Len())
24                  copy(buf.parent.files[buf.name], buf.writer.Bytes())
25                  pool.PutBytesBuffer(buf.writer)
26                  buf.writer = nil
27          }
28          buf.reader = nil
29          return nil
30  }
31
32  func (buf *bufCloser) Write(p []byte) (int, error) {
33          return buf.writer.Write(p)
34  }
35
36  func (buf *bufCloser) Seek(pos int64, whence int) (int64, error) {
37          return buf.reader.Seek(pos, whence)
38  }
39
40  func (buf *bufCloser) Read(b []byte) (int, error) {
41          return buf.reader.Read(b)
42  }
43
44  func (buf *bufCloser) WriteTo(w io.Writer) (n int64, err error) {
45          return buf.reader.WriteTo(w)
```

```
}

type memoryIterator struct {
        files  []string
        prefix string
        c      int
        lock   sync.Mutex
}

func (it *memoryIterator) Next() (string, error) {
        it.lock.Lock()
        defer it.lock.Unlock()
        var actual string
        for !strings.HasPrefix(actual, it.prefix) || actual == "" {
                if len(it.files) <= it.c {
                        return "", errDone
                }
                actual = it.files[it.c]
                it.c++
        }
        return actual, nil
}

type memoryManager struct {
        files map[string][]byte
        lock  sync.RWMutex
}

func (man *memoryManager) Reader(file string) (ReaderSeekerCloser, error)
↪ {
        man.lock.RLock()
        defer man.lock.RUnlock()
        bs, ok := man.files[file]

        if !ok {
                return nil, newFileError(file, errNotExist)
        }
        return &bufCloser{
                reader: bytes.NewReader(bs),
```

```
84                  name:   file,
85          }, nil
86  }
87  func (man *memoryManager) Writer(file string) (io.WriteCloser, error) {
88          man.lock.Lock()
89          defer man.lock.Unlock()
90          if file == "" {
91                  return nil, newFileError(file, errEmpty)
92          }
93          man.files[file] = nil
94          return &bufCloser{
95                  writer: pool.GetBytesBuffer(),
96                  parent: man,
97                  name:   file,
98          }, nil
99  }
100
101 func (man *memoryManager) Delete(file string) error {
102         man.lock.Lock()
103         defer man.lock.Unlock()
104         var err error
105         if file == "" {
106                 err = errEmpty
107         }
108         if _, ok := man.files[file]; !ok && err == nil {
109                 err = errNotExist
110         }
111         if err == nil {
112                 delete(man.files, file)
113         }
114         return newFileError(file, err)
115 }
116
117 func (man *memoryManager) Iterator(prefix string) (Iterator, error) {
118         man.lock.RLock()
119         defer man.lock.RUnlock()
120         var files []string
121         for file := range man.files {
122                 files = append(files, file)
```

```
123         }
124         return &memoryIterator{
125                 files: files,
126         }, nil
127 }
128
129 // NewMemoryManager returns a Manager that only stores buffers in memory
130 func NewMemoryManager() Manager {
131         return &memoryManager{
132                 files: make(map[string][]byte, 0),
133         }
134 }
```

Arquivo multi.go

```
1  package file
2
3  import (
4          "io"
5          "sync"
6  )
7
8  type multiIterator struct {
9          iterators []Iterator
10         lock      sync.Mutex
11 }
12
13 func (it *multiIterator) Next() (string, error) {
14         it.lock.Lock()
15         defer it.lock.Unlock()
16         var actual string
17         err := errDone
18         if len(it.iterators) > 0 {
19                 actual, err = it.iterators[0].Next()
20                 for IsDoneError(err) && len(it.iterators) > 0 {
21                         it.iterators = it.iterators[1:]
22                         if len(it.iterators) > 0 {
23                                 actual, err = it.iterators[0].Next()
24                         }
25                 }
```

```
26          }
27          return actual, err
28  }
29
30  type multiManager struct {
31          writer  Manager
32          readers []Manager
33          lock    sync.Mutex
34  }
35
36  func (mm *multiManager) Reader(file string) (ReaderSeekerCloser, error) {
37          mm.lock.Lock()
38          defer mm.lock.Unlock()
39          var reader ReaderSeekerCloser
40          err := errNotExist
41          for i := 0; IsNotExistError(err) && i < len(mm.readers); i++ {
42                  manager := mm.readers[i]
43                  reader, err = manager.Reader(file)
44          }
45          return reader, err
46  }
47
48  func (mm *multiManager) Writer(file string) (io.WriteCloser, error) {
49          mm.lock.Lock()
50          defer mm.lock.Unlock()
51          return mm.writer.Writer(file)
52  }
53
54  func (mm *multiManager) Delete(file string) error {
55          mm.lock.Lock()
56          defer mm.lock.Unlock()
57          return mm.writer.Delete(file)
58  }
59
60  func (mm *multiManager) Iterator(prefix string) (Iterator, error) {
61          mm.lock.Lock()
62          defer mm.lock.Unlock()
63          var iterators []Iterator
64          var err error
```

```
65          for _, reader := range mm.readers {
66                  var iterator Iterator
67                  if err == nil {
68                          iterator, err = reader.Iterator(prefix)
69                  }
70                  if err == nil {
71                          iterators = append(iterators, iterator)
72                  }
73          }
74          var result Iterator
75          if err == nil {
76                  result = &multiIterator{
77                          iterators: iterators,
78                  }
79          }
80          return result, err
81  }
82
83  func NewMultiManager(writer Manager, readers []Manager) Manager {
84          return &multiManager{
85                  writer:  writer,
86                  readers: readers,
87          }
88  }
```

B.1.26 Pasta util/indexer

Arquivo base.go

```
1   package indexer
2
3   import (
4           "errors"
5
6           "github.com/RoaringBitmap/roaring"
7           "github.com/fjorgemota/gurudamatricula/util/coder"
8           hashids "github.com/speps/go-hashids"
9   )
10
11  var (
```

```
12          // errDone is returned when there are no more results to return
13          errDone = errors.New("indexer: Query has no more results")
14          // errConflict is returned when a conflict is found when
            ↪  commiting the index
15          errConflict = errors.New("indexer: Conflict detected when
            ↪  commiting update")
16          // errCannotDelete is returned when a delete is registered for a
            ↪  item set
17          // in the same session
18          errCannotDelete = errors.New("indexer: Cannot delete specified
            ↪  record")
19          // ErrNotFound is returned when a item is not found on the
            ↪  controller
20          errNotFound       = errors.New("indexer: not found")
21          errBitmapNotFound = errors.New("indexer: Internal error - Bitmap
            ↪  not found")
22          errBitmapColision = errors.New("indexer: Internal error - Found
            ↪  bitmap colision")
23  )
24
25  const metaKey = "meta"
26
27  // SearchOptions has options related specifically to search
28  type SearchOptions struct {
29          Version string
30          Before  uint32
31          After   uint32
32          Limit   uint32
33  }
34
35  // QueryInfo has data about the page the query is requesting
36  type QueryInfo struct {
37          Version string
38          Last    uint32
39          First   uint32
40          Count   uint64
41  }
42
43  // PageInfo has data about the page the user is requesting
```

```
44 type PageInfo struct {
45         After  uint32
46         Before uint32
47 }
48
49 // IndexIteratorInfo returns additional information about the iterator
50 type IndexIteratorInfo struct {
51         Page  PageInfo
52         Query QueryInfo
53 }
54
55 // IndexIterator allows the user to iterate for all the results of the
   ↪  index
56 type IndexIterator interface {
57         Info() IndexIteratorInfo
58         Next(result interface{}) error
59 }
60
61 // Batch allows to save values directly in the index
62 type Batch interface {
63         Delete(itemKey string) error
64         Update(values map[string]string, itemKey string, item
           ↪  interface{}) error
65         Set(value string, itemKey string, item interface{}) error
66         SetField(field, value string, itemKey string, item interface{})
           ↪  error
67         Commit() error
68 }
69
70 // ExpressionData helps Expression to get data from an index
71 type ExpressionData interface {
72         Get(value string) (*roaring.Bitmap, error)
73         GetField(field, value string) (*roaring.Bitmap, error)
74         GetAll() (*roaring.Bitmap, error)
75 }
76
77 // Expression is a helper that defines the results which should be
   ↪  returned
78 // for the query
```

```
79  type Expression interface {
80          EvaluateBitmap(data ExpressionData) (*roaring.Bitmap, error)
81  }
82
83  // InputDataMeta represents the data that will be sent to a Inserter
84  // so it can create a DataMeta representation of the metadata of
85  // a register in the index
86  type InputDataMeta struct {
87          Key     string
88          Counter uint32
89          Deleted bool
90          Keys    *roaring.Bitmap
91  }
92
93  // DataMeta represents the data saved about a itemKey in the index
94  type DataMeta struct {
95          Counter uint32
96          Deleted bool
97          Keys    []MapKey
98  }
99
100 type TreeEdge struct {
101         Key   rune
102         Value uint32
103 }
104
105 type TreeNode struct {
106         ID    uint32
107         Final bool
108         Edges []TreeEdge
109 }
110
111 // Inserter is a inserter that allows to insert meta-data from the index
    ↪  in
112 // batches - while does not knowing anything about the underlying storage
113 type Inserter interface {
114         AddData(counter uint32, data interface{}) error
115         CommitData() error
116         CommitTree(name string, nodes []TreeNode) error
```

```
117         CommitMap(bitmaps map[string]Bitmap, maps map[MapKey]Map) error
118         CommitDataMeta(maps map[MapKey]Map, dataMetaKeys
            ↪ map[string]InputDataMeta) error
119 }
120
121 // ContextBase allows to get version and manage temporary values on the
    ↪ context
122 type ContextGetter interface {
123         GetVersion() string
124         Get(key string) string
125         CacheSet(key string, value interface{})
126         CacheGet(key string) (interface{}, bool)
127 }
128
129 // ContextSetter allows to set persistent values on the context
130 type ContextSetter interface {
131         ContextGetter
132         Set(key, value string)
133 }
134
135 type Optimizer interface {
136         HasMapKey(ctx ContextGetter, key MapKey) bool
137         HasDataMeta(ctx ContextGetter, key string) bool
138         IsDeleted(counter uint32) bool
139         AddDataMeta(key string) error
140         AddMapKey(key MapKey) error
141         RemoveDeletedKeys(bitmap *roaring.Bitmap) (*roaring.Bitmap,
            ↪ error)
142         Commit() error
143 }
144
145 // Controller allows to manage all the data that is not meta for the
    ↪ index
146 type Controller interface {
147         Insert(ctx ContextSetter) Inserter
148         Optimize(sources []ContextGetter, handler Optimizer, target
            ↪ ContextSetter) error
149         GetMap(ctx ContextGetter, key MapKey, target *roaring.Bitmap)
            ↪ error
```

```
150         GetTreeNode(ctx ContextGetter, tree string, node uint32, target
            ↪  *TreeNode) error
151         GetDataMeta(ctx ContextGetter, itemKey string, target *DataMeta)
            ↪  error
152         GetData(ctx ContextGetter, counter uint32, data interface{})
            ↪  error
153         Delete(ctx ContextGetter, toPreserve bool, values []string) error
154 }
155
156 // Index works with a kv.Store to index data so queries is much more
    ↪  faster
157 // and flexible
158 type Index interface {
159         Update() (Batch, error)
160         Rebuild() (Batch, error)
161         Optimize() error
162         Delete() error
163         RunGC() error
164         Search(expression Expression) IndexIterator
165         SearchWithOptions(expression Expression, options SearchOptions)
            ↪  IndexIterator
166 }
167
168 var emptyBitmap *roaring.Bitmap
169 var defaultEncoder coder.IntEncoder
170 var globalErr error
171
172 func init() {
173         emptyBitmap = roaring.New()
174         data := hashids.NewData()
175         bs := make([]byte, 0, 50)
176         for c := byte('0'); c < byte('9'); c++ {
177                 bs = append(bs, c)
178         }
179         for c := byte('a'); c < byte('z'); c++ {
180                 bs = append(bs, c)
181         }
182         data.Alphabet = string(bs)
183         data.Salt = "indexer package"
```

```
184         var defaultHasher *hashids.HashID
185         defaultHasher, globalErr = hashids.NewWithData(data)
186         defaultEncoder = coder.NewHashIDEncoder(defaultHasher)
187 }
```

Arquivo controller.go

```
1  package indexer
2
3  import (
4          "bytes"
5          "encoding/hex"
6          "io"
7          "strings"
8
9          "github.com/RoaringBitmap/roaring"
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11         "github.com/fjorgemota/gurudamatricula/util/kv"
12         "github.com/fjorgemota/gurudamatricula/util/pool"
13 )
14
15 // indexMapItem is used just to host data related to an Indexer instance
   ↪  in the
16 // Store
17 type indexMapItem struct {
18         Field string `datastore:"field,noindex" json:"field"`
19         Value string `datastore:"value,noindex" json:"value"`
20         Items []byte `datastore:"items,noindex" json:"items"`
21 }
22
23 type indexTreeEdge struct {
24         Key   rune  `datastore:"key,noindex" json:"key"`
25         Value int32 `datastore:"value,noindex" json:"value"`
26 }
27
28 type indexTreeNode struct {
29         Node  int32           `datastore:"node,noindex" json:"node"`
30         Final bool            `datastore:"final,noindex" json:"final"`
31         Edges []indexTreeEdge `datastore:"edges,noindex" json:"edges"`
32 }
```

```
33
34 // indexData is used just to host data related to an Indexer instance in
   ↪  the
35 // Store
36 type indexData struct {
37         Counter int32  `datastore:"counter,noindex" json:"counter"`
38         Data    []byte `datastore:"data,noindex" json:"data"`
39 }
40
41 // indexDataMeta is used just to host data related to an Indexer instance
   ↪  in the
42 // Store
43 type indexDataMeta struct {
44         Counter int32    `datastore:"counter,noindex" json:"counter"`
45         Deleted bool     `datastore:"deleted,noindex" json:"deleted"`
46         Name    string   `datastore:"name,noindex" json:"name"`
47         Keys    []MapKey `datastore:"keys,noindex" json:"keys"`
48 }
49
50 func kvEncoder(key MapKey) string {
51         hasher := pool.GetSha1()
52         _, err := io.WriteString(hasher, key.Field)
53         if err == nil {
54                 _, err = io.WriteString(hasher, "-")
55         }
56         if err == nil {
57                 _, err = io.WriteString(hasher, key.Value)
58         }
59         var tmp [20]byte
60         bs := hasher.Sum(tmp[:0])
61         pool.PutSha1(hasher)
62         return hex.EncodeToString(bs)
63 }
64
65 type KVOptions struct {
66         TablePrefix string
67         Coder       coder.Coder
68         IntEncoder  coder.IntEncoder
69         Encoder     func(MapKey) string
```

```
70  }
71
72  func getDefaultKVOptions(options KVOptions) KVOptions {
73          if options.Encoder == nil {
74                  options.Encoder = kvEncoder
75          }
76          if options.Coder == nil {
77                  options.Coder = coder.NewGobCoder()
78          }
79          if options.IntEncoder == nil {
80                  options.IntEncoder = defaultEncoder
81          }
82          return options
83  }
84
85  type kvTables struct {
86          mapTable      string
87          treeTable     string
88          dataMetaTable string
89          dataTable     string
90  }
91
92  type kvController struct {
93          tables  kvTables
94          storage kv.Store
95          options KVOptions
96  }
97
98  func (kc kvController) getDataKey(version string, counter uint32) (string,
    ↪  error) {
99          hash, err := kc.options.IntEncoder.Encode(int(counter))
100         var result string
101         if err == nil {
102                 result = version + "-" + hash
103         }
104         return result, err
105 }
106 func (kc kvController) getMapKey(version string, key MapKey) string {
107         item := kc.options.Encoder(key)
```

```
        return version + "-" + item
}

func (kc kvController) getTreeKey(version, treeName string, nodeID
↪  uint32) (string, error) {
        hash, err := kc.options.IntEncoder.Encode(int(nodeID))
        var result string
        if err == nil {
                result = version + "-" + treeName + "-" + hash
        }
        return result, err
}

func (kc kvController) getDataMetaKey(version string, key string) string
↪ {
        return version + "-" + key
}

func (kc kvController) Insert(ctx ContextSetter) Inserter {
        return &kvInserter{
                dataBuf:    pool.GetBytesBuffer(),
                controller: kc,
                version:    ctx.GetVersion(),
        }
}

func (kc kvController) optimizeMap(sources []ContextGetter, handler
↪  Optimizer, target ContextSetter, known map[string]struct{}) error {
        it := kc.storage.GetAllKeys(kc.tables.mapTable)
        var err error
        for err == nil {
                var key string
                var data indexMapItem
                inSource := -1
                key, err = it.Next(&data)
                var encoded string
                if err == nil {
                        for sourceID, source := range sources {
                                version := source.GetVersion()
```

```
144                             if strings.HasPrefix(key, version) {
145                                     inSource = sourceID
146                                     encoded = key[len(version):]
147                                     break
148                             }
149                     }
150             }
151             if inSource > -1 && err == nil {
152                     // We need to load the key we found here because
                        ↪  we need the MapKey data
153                     if _, ok := known[encoded]; ok {
154                             // We already migrated that MapKey!
155                             // Ignore it here. :)
156                             continue
157                     }
158                     known[encoded] = zero
159                     err = kc.storage.Get(kc.tables.mapTable, key,
                        ↪  &data)
160             }
161             if inSource > -1 && err == nil {
162                     mapKey := MapKey{
163                             Field: data.Field,
164                             Value: data.Value,
165                     }
166                     resultBitmap := roaring.New()
167                     _, err = resultBitmap.FromBuffer(data.Items)
168                     newBitmap := roaring.New()
169                     for sourceID, source := range sources {
170                             if err == nil && sourceID != inSource &&
                                ↪  handler.HasMapKey(source, mapKey) {
171                                     err = kc.GetMap(source, mapKey,
                                        ↪  newBitmap)
172                                     if IsNotFoundError(err) {
173                                             err = nil
174                                     } else if err == nil {
175                                             // Checked! Load bitmap
176                                             resultBitmap.Or(newBitmap)
177                                     }
178                             }
```

```
                        }
                        var bs []byte
                        if err == nil {
                                resultBitmap, err =
                                ↪  handler.RemoveDeletedKeys(resultBitmap)
                                if err == nil && resultBitmap.IsEmpty() {
                                        // Ignore if empty because we
                                        ↪  don't migrate deleted data
                                        continue
                                }
                                if err == nil {
                                        resultBitmap.RunOptimize()
                                        bs, err = resultBitmap.ToBytes()
                                }
                        }
                        if err == nil {
                                newData := indexMapItem{
                                        Field: mapKey.Field,
                                        Value: mapKey.Value,
                                        Items: bs,
                                }
                                newKey :=
                                ↪  kc.getMapKey(target.GetVersion(),
                                ↪  mapKey)
                                err = kc.storage.Set(kc.tables.mapTable,
                                ↪  newKey, &newData)
                        }
                        if err == nil {
                                err = handler.AddMapKey(mapKey)
                        }
                }
        }
        if kv.IsDoneError(err) {
                err = nil
        }
        return err
}

```

```
212 func (kc kvController) optimizeDataMeta(sources []ContextGetter, handler
    ↪ Optimizer, target ContextSetter, known map[string]struct{}) error {
213         it := kc.storage.GetAllKeys(kc.tables.dataMetaTable)
214         var err error
215         for err == nil {
216                 var key string
217                 var data indexDataMeta
218                 var inSource bool
219                 var itemKey string
220                 key, err = it.Next(&data)
221                 if err == nil {
222                         for _, source := range sources {
223                                 version := source.GetVersion()
224                                 if strings.HasPrefix(key, version) {
225                                         inSource = true
226                                         itemKey = key[len(version):]
227                                         break
228                                 }
229                         }
230                 }
231                 if inSource && err == nil {
232                         // We need to load the key we found here because
                        ↪ we need the MapKey data
233                         if _, ok := known[itemKey]; ok {
234                                 // We already migrated that DataMeta!
235                                 // Ignore it here. :)
236                                 continue
237                         }
238                         known[itemKey] = zero
239                         err = kc.storage.Get(kc.tables.dataMetaTable, key,
                        ↪ &data)
240                 }
241                 if inSource && err == nil {
242                         newKey := kc.getDataMetaKey(target.GetVersion(),
                        ↪ data.Name)
243                         var oldData DataMeta
244                         for _, source := range sources {
```

```
245                                 if err == nil &&
                                    ↪  handler.HasDataMeta(source,
                                    ↪  data.Name) {
246                                         err = kc.GetDataMeta(source,
                                            ↪  data.Name, &oldData)
247                                         if IsNotFoundError(err) {
248                                                 err = nil
249                                         } else if err == nil {
250                                                 // Save same data under
                                                    ↪  new Key, if it's not
                                                    ↪  deleted, of course
251                                                 if !oldData.Deleted {
252                                                         err =
                                                            ↪  kc.storage.Set(kc.tables.dat
                                                            ↪  newKey,
                                                            ↪  &indexDataMeta{
253                                                                 Counter:
                                                                    ↪  int32(oldData.Counte
254                                                                 Deleted:
                                                                    ↪  oldData.Deleted,
255                                                                 Keys:
                                                                    ↪  oldData.Keys,
256                                                                 Name:
                                                                    ↪  data.Name,
257                                                         })
258                                                         if err == nil {
259                                                                 err =
                                                                    ↪  handler.AddDataMeta(
260                                                         }
261                                                 }
262                                                 break
263                                         }
264                                 }
265                         }
266                 }
267         }
268         if kv.IsDoneError(err) {
269                 err = nil
270         }
```

```
271         return err
272 }
273 func (kc kvController) optimizeData(sources []ContextGetter, handler
    ↪  Optimizer, target ContextSetter) error {
274         it := kc.storage.GetAllKeys(kc.tables.dataTable)
275         var err error
276         for err == nil {
277                 var key string
278                 var data indexData
279                 var inSource bool
280                 key, err = it.Next(&data)
281                 if err == nil {
282                         for _, source := range sources {
283                                 version := source.GetVersion()
284                                 if strings.HasPrefix(key, version) {
285                                         inSource = true
286                                         break
287                                 }
288                         }
289                 }
290                 if inSource && err == nil {
291                         err = kc.storage.Get(kc.tables.dataTable, key,
                        ↪  &data)
292                 }
293                 if inSource && err == nil &&
                ↪  !handler.IsDeleted(uint32(data.Counter)) {
294                         var newKey string
295                         newKey, err = kc.getDataKey(target.GetVersion(),
                        ↪  uint32(data.Counter))
296                         // Save same data under new Key
297                         if err == nil {
298                                 err = kc.storage.Set(kc.tables.dataTable,
                                ↪  newKey, &data)
299                         }
300                 }
301         }
302         if kv.IsDoneError(err) {
303                 err = nil
304         }
```

```
305         return err
306 }
307
308 func (kc kvController) Optimize(sources []ContextGetter, handler
    ↪  Optimizer, target ContextSetter) error {
309         known := mapPool.Get().(map[string]struct{})
310         err := kc.optimizeMap(sources, handler, target, known)
311         for key := range known {
312                 delete(known, key)
313         }
314         if err == nil {
315                 err = kc.optimizeDataMeta(sources, handler, target,
                    ↪  known)
316         }
317         for key := range known {
318                 delete(known, key)
319         }
320         if err == nil {
321                 err = kc.optimizeData(sources, handler, target)
322         }
323         if err == nil {
324                 err = handler.Commit()
325         }
326         mapPool.Put(known)
327         return err
328 }
329
330 func (kc kvController) GetMap(ctx ContextGetter, key MapKey, bitmap
    ↪  *roaring.Bitmap) error {
331         dbKey := kc.getMapKey(ctx.GetVersion(), key)
332         var data indexMapItem
333         err := kc.storage.Get(kc.tables.mapTable, dbKey, &data)
334         if err == nil && data.Field == key.Field && data.Value ==
            ↪  key.Value {
335                 _, err = bitmap.FromBuffer(data.Items)
336         } else if err == nil || kv.IsKeyNotFoundError(err) {
337                 err = errNotFound
338         }
339         return err
```

```
340
341 }
342 func (kc kvController) GetDataMeta(ctx ContextGetter, itemKey string,
    ↪  result *DataMeta) error {
343         dbKey := kc.getDataMetaKey(ctx.GetVersion(), itemKey)
344         var data indexDataMeta
345         err := kc.storage.Get(kc.tables.dataMetaTable, dbKey, &data)
346         if err == nil && data.Name == itemKey {
347                 result.Counter = uint32(data.Counter)
348                 result.Deleted = data.Deleted
349                 result.Keys = append(result.Keys[:0], data.Keys...)
350         } else if err == nil || kv.IsKeyNotFoundError(err) {
351                 err = errNotFound
352         }
353         return err
354 }
355
356 func (kc kvController) GetTreeNode(ctx ContextGetter, name string, nodeID
    ↪  uint32, result *TreeNode) error {
357         key, err := kc.getTreeKey(ctx.GetVersion(), name, nodeID)
358         var dbResult indexTreeNode
359         var ok bool
360         if err == nil {
361                 var tmp interface{}
362                 tmp, ok = ctx.CacheGet(key)
363                 if ok {
364                         dbResult, ok = tmp.(indexTreeNode)
365                 }
366         }
367         if err == nil && !ok {
368                 err = kc.storage.Get(kc.tables.treeTable, key, &dbResult)
369         }
370         if err == nil && dbResult.Node == int32(nodeID) {
371                 if !ok {
372                         ctx.CacheSet(key, dbResult)
373                 }
374                 result.ID = nodeID
375                 result.Final = dbResult.Final
376                 edgesLen := len(dbResult.Edges)
```

```
377                 resultCap := cap(result.Edges)
378                 if resultCap < edgesLen {
379                         result.Edges = append(result.Edges,
                            ↪ make([]TreeEdge,
                            ↪ edgesLen-len(result.Edges))...)
380                 }
381                 result.Edges = result.Edges[:edgesLen]
382                 for edgeID, edge := range dbResult.Edges {
383                         result.Edges[edgeID] = TreeEdge{
384                                 Key:   edge.Key,
385                                 Value: uint32(edge.Value),
386                         }
387                 }
388         } else if err == nil || kv.IsKeyNotFoundError(err) {
389                 err = errNotFound
390         }
391         return err
392 }
393
394 func (kc kvController) GetData(ctx ContextGetter, counter uint32, data
    ↪ interface{}) error {
395         key, err := kc.getDataKey(ctx.GetVersion(), counter)
396         var result indexData
397         if err == nil {
398                 err = kc.storage.Get(kc.tables.dataTable, key, &result)
399         }
400         if err == nil {
401                 reader := bytes.NewReader(result.Data)
402                 err = kc.options.Coder.DecodeFrom(reader, data)
403         }
404         return err
405 }
406
407 func (kc kvController) deleteTable(ctx ContextGetter, table string,
    ↪ result interface{}, toPreserve bool, values []string) error {
408         var err error
409         it := kc.storage.GetAllKeys(table)
410         for err == nil {
411                 var key string
```

```
412                 key, err = it.Next(result)
413                 if err == nil && strings.HasPrefix(key, ctx.GetVersion())
                    ↪ {
414                         hasPrefix := false
415                         for _, check := range values {
416                                 if strings.HasPrefix(key, check) {
417                                         hasPrefix = true
418                                         break
419                                 }
420                         }
421                         if !hasPrefix == toPreserve {
422                                 err = kc.storage.Del(table, key)
423                         }
424                 }
425         }
426         if kv.IsDoneError(err) {
427                 err = nil
428         }
429         return err
430 }
431
432 func (kc kvController) Delete(ctx ContextGetter, toPreserve bool, values
    ↪  []string) error {
433         // Used by Delete and RunGC methods in Index
434         err := kc.deleteTable(ctx, kc.tables.mapTable, &indexMapItem{},
            ↪  toPreserve, values)
435         if err == nil {
436                 err = kc.deleteTable(ctx, kc.tables.treeTable,
                    ↪  &indexTreeNode{}, toPreserve, values)
437         }
438         if err == nil {
439                 err = kc.deleteTable(ctx, kc.tables.dataMetaTable,
                    ↪  &indexDataMeta{}, toPreserve, values)
440         }
441         if err == nil {
442                 err = kc.deleteTable(ctx, kc.tables.dataTable,
                    ↪  &indexData{}, toPreserve, values)
443         }
444         return err
```

```
445 }
446
447 func NewKVController(storage kv.Store, options KVOptions) Controller {
448         options = getDefaultKVOptions(options)
449         return kvController{
450                 storage: storage,
451                 options: options,
452                 tables: kvTables{
453                         dataMetaTable: options.TablePrefix +
                        ↪  "-data-meta",
454                         dataTable:     options.TablePrefix + "-data",
455                         mapTable:      options.TablePrefix + "-map",
456                         treeTable:     options.TablePrefix + "-tree",
457                 },
458         }
459 }
```

Arquivo controller_inserter.go

```
1  package indexer
2
3  import (
4          "bytes"
5          "sync"
6
7          "github.com/fjorgemota/gurudamatricula/util/pool"
8  )
9
10 const kvBatchLimit = 32
11
12 type kvDataBatchItem struct {
13         Counter uint32
14         Start   int
15         End     int
16 }
17
18 type kvMapBatchItem struct {
19         Start int
20         End   int
21 }
```

```
22
23 type kvInserter struct {
24         dataBatch      [kvBatchLimit]kvDataBatchItem
25         dataBatchCount int
26         dataBuf        *bytes.Buffer
27         version        string
28         controller     kvController
29         lock           sync.Mutex
30         err            error
31 }
32
33 func (ki *kvInserter) AddData(counter uint32, data interface{}) error {
34         ki.lock.Lock()
35         defer ki.lock.Unlock()
36         result := kvDataBatchItem{
37                 Counter: counter,
38                 Start:   ki.dataBuf.Len(),
39         }
40         if ki.err == nil {
41                 ki.err = ki.controller.options.Coder.EncodeTo(data,
                   ↪  ki.dataBuf)
42         }
43         if ki.err == nil {
44                 result.End = ki.dataBuf.Len()
45                 ki.dataBatch[ki.dataBatchCount] = result
46                 ki.dataBatchCount++
47                 if ki.dataBatchCount == kvBatchLimit {
48                         ki.err = ki.commitData()
49                 }
50         }
51         return ki.err
52 }
53
54 func (ki *kvInserter) commitData() error {
55         batch := ki.controller.storage.Batch()
56         bs := ki.dataBuf.Bytes()
57         for ki.err == nil && ki.dataBatchCount > 0 {
58                 var key string
59                 ki.dataBatchCount--
```

```
60                  item := ki.dataBatch[ki.dataBatchCount]
61                  if ki.err == nil {
62                          key, ki.err = ki.controller.getDataKey(ki.version,
                            ↪  item.Counter)
63                  }
64                  if ki.err == nil {
65                          ki.err = batch.Set(ki.controller.tables.dataTable,
                            ↪  key, &indexData{
66                                  Counter: int32(item.Counter),
67                                  Data:    bs[item.Start:item.End],
68                          })
69                  }
70          }
71          if ki.err == nil {
72                  ki.err = batch.Commit()
73          }
74          ki.dataBuf.Reset()
75          return ki.err
76  }
77
78  func (ki *kvInserter) CommitData() error {
79          ki.lock.Lock()
80          defer ki.lock.Unlock()
81          if ki.err == nil {
82                  ki.err = ki.commitData()
83          }
84          return ki.err
85  }
86
87  func (ki *kvInserter) CommitMap(bitmaps map[string]Bitmap, keys
    ↪  map[MapKey]Map) error {
88          ki.lock.Lock()
89          defer ki.lock.Unlock()
90          bufBitmaps := make(map[string]kvMapBatchItem, len(bitmaps))
91          batch := ki.controller.storage.Batch()
92          for key, value := range bitmaps {
93                  if value.Count > 0 {
94                          entry := kvMapBatchItem{
95                                  Start: ki.dataBuf.Len(),
```

```
96                          }
97                          if ki.err == nil {
98                                  _, ki.err =
                                    ↪  value.Bitmap.WriteTo(ki.dataBuf)
99                          }
100                         if ki.err == nil {
101                                 entry.End = ki.dataBuf.Len()
102                                 bufBitmaps[key] = entry
103                         }
104                 }
105         }
106         bs := ki.dataBuf.Bytes()
107         for key, value := range keys {
108                 if ki.err == nil {
109                         encodedKey := ki.controller.getMapKey(ki.version,
                        ↪  key)
110                         bitmap := bufBitmaps[value.Bitmap]
111                         ki.err = batch.Set(ki.controller.tables.mapTable,
                        ↪  encodedKey, &indexMapItem{
112                                 Items: bs[bitmap.Start:bitmap.End],
113                                 Field: key.Field,
114                                 Value: key.Value,
115                         })
116                 }
117         }
118         if ki.err == nil {
119                 ki.err = batch.Commit()
120         }
121         pool.PutBytesBuffer(ki.dataBuf)
122         ki.dataBuf = nil
123         return ki.err
124 }
125
126 func (ki *kvInserter) CommitTree(name string, tree []TreeNode) error {
127         ki.lock.Lock()
128         defer ki.lock.Unlock()
129         batch := ki.controller.storage.Batch()
130         for nodeID, node := range tree {
131                 var encodedKey string
```

```
132                 if ki.err == nil {
133                         encodedKey, ki.err =
                            ↪ ki.controller.getTreeKey(ki.version, name,
                            ↪ uint32(nodeID))
134                 }
135                 if ki.err == nil {
136                         edges := make([]indexTreeEdge, len(node.Edges))
137                         for edgeID, edge := range node.Edges {
138                                 edges[edgeID] = indexTreeEdge{
139                                         Key:   edge.Key,
140                                         Value: int32(edge.Value),
141                                 }
142                         }
143                         ki.err = batch.Set(ki.controller.tables.treeTable,
                            ↪ encodedKey, &indexTreeNode{
144                                 Node:  int32(nodeID),
145                                 Edges: edges,
146                                 Final: node.Final,
147                         })
148                 }
149         }
150         if ki.err == nil {
151                 ki.err = batch.Commit()
152         }
153         return ki.err
154 }
155
156 func (ki *kvInserter) CommitDataMeta(keys map[MapKey]Map, dataMetaKeys
    ↪ map[string]InputDataMeta) error {
157         ki.lock.Lock()
158         defer ki.lock.Unlock()
159         batch := ki.controller.storage.Batch()
160         keyList := make([]MapKey, len(keys))
161         for key, value := range keys {
162                 keyList[value.Index] = key
163         }
164         for itemKey, meta := range dataMetaKeys {
165                 if ki.err == nil {
```

```
166                         dbKey := ki.controller.getDataMetaKey(ki.version,
                            ↪  itemKey)
167                         var result indexDataMeta
168                         result.Name = itemKey
169                         result.Counter = int32(meta.Counter)
170                         result.Deleted = meta.Deleted
171                         result.Keys = make([]MapKey, 0,
                            ↪  meta.Keys.GetCardinality())
172                         it := meta.Keys.Iterator()
173                         for it.HasNext() {
174                                 index := it.Next()
175                                 key := keyList[index]
176                                 result.Keys = append(result.Keys, key)
177                         }
178                         ki.err =
                            ↪  batch.Set(ki.controller.tables.dataMetaTable,
                            ↪  dbKey, &result)
179                 }
180         }
181         if ki.err == nil {
182                 ki.err = batch.Commit()
183         }
184         return ki.err
185 }
```

Arquivo errors.go

```
1  package indexer
2
3  import "github.com/pkg/errors"
4
5  // IsDoneError checks if the original error was caused by the iterator
   ↪  not
6  // returning any more results
7  func IsDoneError(err error) bool {
8          return errors.Cause(err) == errDone
9  }
10
11 // IsConflictError checks if the original error was caused by a conflict
12 // while saving the index
```

```
13 func IsConflictError(err error) bool {
14         return errors.Cause(err) == errConflict
15 }
16
17 // IsCannotDeleteError checks if the original error was caused by a
   ↪  delete
18 // after a set on the indexing process
19 func IsCannotDeleteError(err error) bool {
20         return errors.Cause(err) == errCannotDelete
21 }
22
23 // IsCannotDeleteError checks if the original error was caused by a key
   ↪  that
24 // was not found on a controller
25 func IsNotFoundError(err error) bool {
26         return errors.Cause(err) == errNotFound
27 }
```

Arquivo exact_batch.go

```
1  package indexer
2
3  import (
4          "encoding/binary"
5          "hash"
6          "sync"
7
8          "github.com/RoaringBitmap/roaring"
9          "github.com/fjorgemota/gurudamatricula/util/kv"
10         "github.com/fjorgemota/gurudamatricula/util/pool"
11         "github.com/willf/bloom"
12 )
13
14 type Map struct {
15         Bitmap string
16         Index  uint32
17 }
18
19 type Bitmap struct {
20         // Number of entities using
```

```
21          Count  uint32
22          Bitmap *roaring.Bitmap
23  }
24
25  type exactFieldMap struct {
26          Bitmap      string
27          Index       uint32
28          FieldLength uint8
29  }
30
31  type exactBatch struct {
32          ctx           *exactContext
33          inserter      Inserter
34          allItems      *roaring.Bitmap
35          items         *roaring.Bitmap
36          util          *exactBatchUtil
37          bitmapHasher  hash.Hash
38          dataFilters   []*bloom.BloomFilter
39          mapFilters    []*bloom.BloomFilter
40          sources       []ContextGetter
41          old           exactMetaVersions
42          lock          sync.Mutex
43          newItemsCount uint32
44          committed     bool
45          err           error
46          indexer       *exactIndexer
47  }
48
49  func (es *exactBatch) bitmapEncode(bitmap *roaring.Bitmap) (string,
    ↪  error) {
50          var err error
51          it := bitmap.Iterator()
52          if cap(es.util.tmp) < 5 {
53                  size := 5
54                  hashSize := es.bitmapHasher.Size()
55                  if size < hashSize {
56                          size = hashSize
57                  }
58                  es.util.tmp = append(es.util.tmp, make([]byte, size)...)
```

```
59          }
60          es.util.tmp = es.util.tmp[:4]
61          var count uint32
62          for err == nil && it.HasNext() {
63                  index := it.Next()
64                  count++
65                  binary.LittleEndian.PutUint32(es.util.tmp, count)
66                  _, err = es.bitmapHasher.Write(es.util.tmp)
67                  if err == nil {
68                          binary.LittleEndian.PutUint32(es.util.tmp, index)
69                          _, err = es.bitmapHasher.Write(es.util.tmp)
70                  }
71          }
72          var value string
73          if err == nil {
74                  es.util.tmp = es.util.tmp[:0]
75                  es.util.tmp = es.bitmapHasher.Sum(es.util.tmp)
76                  value = es.internBytes(es.util.tmp)
77                  es.bitmapHasher.Reset()
78          }
79          return value, err
80  }
81
82  func (es *exactBatch) bitmapAdd(key string, x uint32) (string, error) {
83          err := errBitmapNotFound
84          if entry, ok := es.util.bitmaps[key]; ok {
85                  err = nil
86                  if entry.Bitmap.CheckedAdd(x) {
87                          var newKey string
88                          newKey, err = es.bitmapEncode(entry.Bitmap)
89                          if err == nil {
90                                  entry.Count--
91                                  if entry.Count > 0 {
92                                          es.util.bitmaps[key] = entry
93                                  } else {
94                                          delete(es.util.bitmaps, key)
95                                          delete(es.util.stringIntern, key)
96                                  }
97                                  key = newKey
```

```
98                                      if newEntry, ok := es.util.bitmaps[key];
                                        ↪  ok {
99                                              err = errBitmapColision
100                                             if
                                                ↪  newEntry.Bitmap.Equals(entry.Bitmap)
                                                ↪  {
101                                                     err = nil
102                                                     newEntry.Count++
103                                                     es.util.bitmaps[key] =
                                                        ↪  newEntry
104                                                     if entry.Count == 0 {
105                                                             entry.Bitmap.Clear()
106                                                     } else {
107                                                             entry.Bitmap.Remove(x)
108                                                     }
109                                             }
110                                     } else if entry.Count > 0 {
111                                             newBitmap := entry.Bitmap.Clone()
112                                             entry.Bitmap.Remove(x)
113                                             es.util.bitmaps[key] = Bitmap{
114                                                     Count:  1,
115                                                     Bitmap: newBitmap,
116                                             }
117                                     } else {
118                                             entry.Count++
119                                             es.util.bitmaps[key] = entry
120                                     }
121                             }
122                     }
123             }
124             return key, err
125     }
126
127     func (es *exactBatch) bitmapInitialize(x uint32) (string, error) {
128             bitmap := roaring.New()
129             bitmap.Add(x)
130             key, err := es.bitmapEncode(bitmap)
131             entry, ok := es.util.bitmaps[key]
132             if !ok {
```

```
133                 entry.Bitmap = bitmap
134         }
135         entry.Count++
136         es.util.bitmaps[key] = entry
137         return key, err
138 }
139
140 func (es *exactBatch) loadOldItem(itemKey string) error {
141         var err error
142         if _, exists := es.util.oldItems[itemKey]; len(es.old.Versions)
        ↪  == 0 || exists {
143                 return err
144         }
145         es.util.oldItems[itemKey] = zero
146         for i, source := range es.sources {
147                 if err == nil && es.dataFilters[i].TestString(itemKey) {
148                         var meta DataMeta
149                         err = es.indexer.controller.GetDataMeta(source,
                        ↪  itemKey, &meta)
150                         if IsNotFoundError(err) {
151                                 // It's a false positive, ignore! :D
152                                 err = nil
153                         } else if err == nil {
154                                 es.util.dataMeta[itemKey] =
                                ↪  InputDataMeta{
155                                         Key:     itemKey,
156                                         Counter: meta.Counter,
157                                         Deleted: meta.Deleted,
158                                         Keys:    roaring.New(),
159                                 }
160                                 for _, key := range meta.Keys {
161                                         if err == nil {
162                                                 var entry Map
163                                                 var ok bool
164                                                 var newBitmap string
165                                                 if entry, ok =
                                                ↪  es.util.keys[key];
                                                ↪  !ok {
```

```
166                                                     newBitmap, err =
                                                        ↪  es.bitmapInitialize(met
167                                                     entry.Index =
                                                        ↪  uint32(len(es.util.keys
168                                                 } else {
169                                                     newBitmap, err =
                                                        ↪  es.bitmapAdd(entry.Bitm
                                                        ↪  meta.Counter)
170                                                 }
171                                                 if newBitmap != "" &&
                                                    ↪  newBitmap !=
                                                    ↪  entry.Bitmap {
172                                                     entry.Bitmap =
                                                        ↪  newBitmap
173                                                     es.util.keys[key]
                                                        ↪  = entry
174                                                 }
175                                                 es.util.dataMeta[itemKey].Keys.Add(
176                                             }
177                                         }
178                                         break
179                                 }
180                         }
181                 }
182                 return err
183         }
184
185         func (es *exactBatch) Delete(itemKey string) error {
186                 es.lock.Lock()
187                 defer es.lock.Unlock()
188                 itemKey = es.intern(itemKey)
189                 if !es.committed && es.err == nil && len(es.old.Versions) > 0 {
190                         es.err = es.loadOldItem(itemKey)
191                         var exists bool
192                         var meta InputDataMeta
193                         if meta, exists = es.util.dataMeta[itemKey]; es.err ==
                                ↪  nil && exists && es.items.Contains(meta.Counter) {
194                                 es.err = errCannotDelete
195                         } else if !exists || meta.Deleted {
```

```
196                         return es.err
197                 }
198                 if es.err == nil {
199                         meta.Deleted = true
200                         meta.Keys.Clear()
201                         es.allItems.Remove(meta.Counter)
202                         es.util.dataMeta[itemKey] = meta
203                 }
204         }
205         return es.err
206 }
207
208 func (es *exactBatch) Update(values map[string]string, itemKey string,
    ↪  item interface{}) error {
209         err := es.Delete(itemKey)
210         for field, value := range values {
211                 if err == nil {
212                         err = es.SetField(field, value, itemKey, item)
213                 }
214         }
215         return err
216 }
217
218 func (es *exactBatch) Set(value string, itemKey string, item interface{})
    ↪  error {
219         return es.SetField(es.indexer.options.DefaultField, value,
            ↪  itemKey, item)
220 }
221
222 func (es *exactBatch) getNextID() uint32 {
223         var min uint32
224         if len(es.old.Versions) > 0 {
225                 // The old max is now the new minimum
226                 min = uint32(es.old.Versions[0].Max)
227         }
228         return es.newItemsCount + min
229 }
230 func (es *exactBatch) internBytes(bs []byte) string {
231         var result string
```

```
232         var ok bool
233         if result, ok = es.util.stringIntern[string(bs)]; !ok {
234                 result = string(bs)
235                 es.util.stringIntern[result] = result
236         }
237         return result
238 }
239
240 func (es *exactBatch) intern(str string) string {
241         var result string
242         var ok bool
243         if result, ok = es.util.stringIntern[str]; !ok {
244                 result = str
245                 es.util.stringIntern[str] = result
246         }
247         return result
248 }
249
250 func (es *exactBatch) SetField(field, value, itemKey string, item
    ↪  interface{}) error {
251         es.lock.Lock()
252         defer es.lock.Unlock()
253         field = es.intern(field)
254         value = es.intern(value)
255         itemKey = es.intern(itemKey)
256         if !es.committed && es.err == nil {
257                 // Load any old content on the bitmap so we do not
                    ↪  overwrite anything
258                 if len(es.old.Versions) > 0 {
259                         es.err = es.loadOldItem(itemKey)
260                 }
261                 key := MapKey{Field: field, Value: value}
262                 if key, keyExist := es.util.dataMeta[itemKey]; !keyExist
                    ↪  || key.Deleted {
263                         es.newItemsCount++
264                         es.util.dataMeta[itemKey] = InputDataMeta{
265                                 Key:     itemKey,
266                                 Counter: es.getNextID(),
267                                 Keys:    roaring.New(),
```

```
268                         }
269                 }
270                 // Key was set! So just mark it as NOT deleted
271                 dataMetaEntry := es.util.dataMeta[itemKey]
272                 counter := dataMetaEntry.Counter
273                 var keyEntry Map
274                 var ok bool
275                 var newBitmap string
276                 if keyEntry, ok = es.util.keys[key]; !ok {
277                         newBitmap, es.err = es.bitmapInitialize(counter)
278                         keyEntry.Index = uint32(len(es.util.keys))
279                 } else if es.err == nil &&
                    ↪  dataMetaEntry.Keys.CheckedAdd(keyEntry.Index) {
280                         newBitmap, es.err = es.bitmapAdd(keyEntry.Bitmap,
                            ↪  counter)
281                 }
282                 if newBitmap != "" && newBitmap != keyEntry.Bitmap {
283                         dataMetaEntry.Keys.Add(keyEntry.Index)
284                         keyEntry.Bitmap = newBitmap
285                         es.util.keys[key] = keyEntry
286                 }
287                 es.util.dataMeta[itemKey] = dataMetaEntry
288                 isNew := es.allItems.CheckedAdd(counter)
289                 es.items.Add(counter)
290                 if isNew && es.err == nil {
291                         es.err = es.inserter.AddData(counter, item)
292                 }
293         }
294         return es.err
295 }
296
297 func (es *exactBatch) Commit() error {
298         err := es.commit(nil)
299         autoMerge := es.indexer.options.AutoMergeMinVersions
300         oldVersionsLen := len(es.old.Versions) + 1
301         if err == nil && autoMerge > 0 && autoMerge > oldVersionsLen {
302                 err = es.indexer.Optimize()
303         }
304         return err
```

```
305 }
306
307 func (es *exactBatch) commit(callback func() error) error {
308         es.lock.Lock()
309         defer es.lock.Unlock()
310         es.committed = true
311         err := es.err
312         if err == nil {
313                 err = es.inserter.CommitData()
314         }
315         if err == nil && callback != nil {
316                 err = callback()
317         }
318         es.allItems.RunOptimize()
319         for key := range es.util.bitmaps {
320                 entry := es.util.bitmaps[key]
321                 entry.Bitmap.And(es.allItems)
322                 entry.Bitmap.RunOptimize()
323                 es.util.bitmaps[key] = entry
324         }
325         mapKeysCount := len(es.util.keys)
326         mapKeys := bloom.NewWithEstimates(uint(mapKeysCount),
          ↪  es.indexer.options.FalsePositiveKeys)
327         for key := range es.util.keys {
328                 es.util.tmp = es.util.tmp[:0]
329                 es.util.tmp = key.Bytes(es.util.tmp)
330                 mapKeys.Add(es.util.tmp)
331         }
332         if err == nil {
333                 err = es.inserter.CommitMap(es.util.bitmaps,
                  ↪  es.util.keys)
334         }
335         dataKeysCount := len(es.util.dataMeta)
336         dataKeys := bloom.NewWithEstimates(uint(dataKeysCount),
          ↪  es.indexer.options.FalsePositiveKeys)
337         for key, value := range es.util.dataMeta {
338                 es.util.tmp = es.util.tmp[:0]
339                 es.util.tmp = append(es.util.tmp, key...)
340                 dataKeys.Add(es.util.tmp)
```

```
341                 value.Keys.RunOptimize()
342         }
343         if err == nil {
344                 err = es.inserter.CommitDataMeta(es.util.keys,
                    ↪  es.util.dataMeta)
345         }
346         // Put exactBatchUtil in pool and clear reference to it
347         putExactBatchUtil(es.util)
348         tmpBuf := pool.GetBytesBuffer()
349         defer pool.PutBytesBuffer(tmpBuf)
350         var encodedAllItems int
351         if err == nil {
352                 es.allItems.RunOptimize()
353                 _, err = es.allItems.WriteTo(tmpBuf)
354                 es.allItems = nil
355                 encodedAllItems = tmpBuf.Len()
356         }
357         var encodedItems int
358         if err == nil {
359                 es.items.RunOptimize()
360                 _, err = es.items.WriteTo(tmpBuf)
361                 es.items = nil
362                 encodedItems = tmpBuf.Len()
363         }
364         var encodedMapKeys int
365         if err == nil {
366                 _, err = mapKeys.WriteTo(tmpBuf)
367                 mapKeys = nil
368                 encodedMapKeys = tmpBuf.Len()
369         }
370         if err == nil {
371                 _, err = dataKeys.WriteTo(tmpBuf)
372                 dataKeys = nil
373         }
374         if err == nil {
375                 // If all went OK, we can update the meta table to have
                    ↪  the right version
376                 metaTable := es.indexer.getMetaTable()
377                 indexMetaKey := es.indexer.getMetaKey()
```

```
378                     bs := tmpBuf.Bytes()
379                     err =
                        ↪  es.indexer.storage.Transaction([]kv.Reference{{Table:
                        ↪  metaTable, Key: indexMetaKey}, {Table: metaTable, Key:
                        ↪  es.ctx.GetVersion()}}, func(db kv.LimitedStore) error
                        ↪  {
380                             var oldMeta, newMeta exactMetaVersions
381                             var metaVersion exactMetaVersion
382                             errOp := db.Get(metaTable, indexMetaKey,
                                ↪  &oldMeta)
383                             if errOp == nil && len(es.old.Versions) > 0 {
384                                     errOp = errConflict
385                                     versions1 := oldMeta.Versions
386                                     versions2 := es.old.Versions
387                                     if len(versions1) == len(versions2) {
388                                             count := 0
389                                             for i := range versions2 {
390                                                     if versions1[i].Version
                                                        ↪  ==
                                                        ↪  versions2[i].Version
                                                        ↪  {
391                                                             count++
392                                                     }
393                                             }
394                                             if count == len(versions2) {
395                                                     errOp = nil
396                                             }
397                                     }
398                             }
399                             metaVersion = exactMetaVersion{
400                                     Version:       es.ctx.GetVersion(),
401                                     AllItems:      bs[:encodedAllItems],
402                                     Items:
                                        ↪  bs[encodedAllItems:encodedItems],
403                                     AllMapKeys:
                                        ↪  bs[encodedItems:encodedMapKeys],
404                                     AllDataKeys:   bs[encodedMapKeys:],
405                                     MapKeysCount:  mapKeysCount,
406                                     DataKeysCount: dataKeysCount,
```

```
407                             Max:           int32(es.getNextID()),
408                     }
409                     newMeta.Versions = append(newMeta.Versions,
                        ↪  metaVersion)
410                     if errOp == nil {
411                             newMeta.Versions =
                                ↪  append(newMeta.Versions,
                                ↪  es.old.Versions...)
412                     }
413                     for key, value := range es.ctx.config {
414                             newMeta.Config = append(newMeta.Config,
                                ↪  exactMetaConfig{
415                                     Version: es.ctx.GetVersion(),
416                                     Key:     key,
417                                     Value:   value,
418                             })
419                     }
420                     newMeta.Config = append(newMeta.Config,
                        ↪  es.old.Config...)
421                     if kv.IsKeyNotFoundError(errOp) {
422                             errOp = nil
423                     }
424                     if errOp == nil {
425                             errOp = db.Set(es.indexer.getMetaTable(),
                                ↪  es.ctx.GetVersion(), &newMeta)
426                     }
427                     if errOp == nil {
428                             errOp = db.Set(es.indexer.getMetaTable(),
                                ↪  indexMetaKey, &newMeta)
429                     }
430                     return errOp
431             })
432     }
433     es.ctx = nil
434     es.dataFilters = nil
435     es.mapFilters = nil
436     es.sources = nil
437     es.util = nil
438     es.inserter = nil
```

```
439         return err
440 }
```

Arquivo exact_context.go

```
1  package indexer
2
3  import (
4          lru "github.com/hashicorp/golang-lru"
5  )
6
7  type exactCache struct {
8          values *lru.Cache
9  }
10
11 func (ec *exactCache) Set(key string, value interface{}) {
12         ec.values.Add(key, value)
13 }
14
15 func (ec *exactCache) Get(key string) (interface{}, bool) {
16         return ec.values.Get(key)
17 }
18
19 func newExactCache(capacity int) (*exactCache, error) {
20         var result *exactCache
21         values, err := lru.New((capacity + 1) * 2)
22         if err == nil {
23                 result = &exactCache{
24                         values: values,
25                 }
26         }
27         return result, nil
28 }
29
30 type exactContext struct {
31         config  map[string]string
32         cache   *exactCache
33         version string
34 }
35
```

```
36 func (ec *exactContext) GetVersion() string {
37         return ec.version
38 }
39
40 func (ec *exactContext) Get(key string) string {
41         return ec.config[key]
42 }
43
44 func (ec *exactContext) Set(key, value string) {
45         ec.config[key] = value
46 }
47
48 func (ec *exactContext) getCacheKey(key string) string {
49         return ec.version + "-" + key
50 }
51
52 func (ec *exactContext) CacheSet(key string, value interface{}) {
53         ec.cache.Set(ec.getCacheKey(key), value)
54 }
55
56 func (ec *exactContext) CacheGet(key string) (interface{}, bool) {
57         return ec.cache.Get(ec.getCacheKey(key))
58 }
```

Arquivo exact_indexer.go

```
1  package indexer
2
3  import (
4          "strings"
5
6          "github.com/RoaringBitmap/roaring"
7          "github.com/fjorgemota/gurudamatricula/util/kv"
8          "github.com/willf/bloom"
9  )
10
11 type exactIndexer struct {
12         storage    kv.Store // t stores data for the index
13         options    ExactOptions
14         controller Controller
```

```
15 }
16
17 func (eIndex *exactIndexer) getMetaTable() string {
18         return eIndex.options.MetaTable
19 }
20
21 func (eIndex *exactIndexer) getMetaKey() string {
22         return eIndex.options.Name + "-" + metaKey
23 }
24
25 func (eIndex *exactIndexer) Rebuild() (Batch, error) {
26         return eIndex.newExactBatch(exactMetaVersions{})
27 }
28
29 func (eIndex *exactIndexer) Update() (Batch, error) {
30         var old exactMetaVersions
31         err := eIndex.storage.Get(eIndex.getMetaTable(),
          ↪  eIndex.getMetaKey(), &old)
32         if kv.IsKeyNotFoundError(err) {
33                 return eIndex.Rebuild()
34         }
35         var batch Batch
36         if err == nil {
37                 batch, err = eIndex.newExactBatch(old)
38         }
39         return batch, err
40 }
41
42 func (eIndex *exactIndexer) generateNewVersionID() (string, error) {
43         version, err := eIndex.storage.GenerateID(eIndex.getMetaTable())
44         if err == nil {
45                 version = eIndex.options.Name + "-" + version
46         }
47         return version, err
48 }
49
50 func (eIndex *exactIndexer) newExactBatch(old exactMetaVersions)
   ↪  (*exactBatch, error) {
51         var version string
```

```
52          err := globalErr
53          if err == nil {
54                  version, err = eIndex.generateNewVersionID()
55          }
56          var saver *exactBatch
57          var dataFilters, mapFilters []*bloom.BloomFilter
58          if err == nil {
59                  dataFilters, err = getDataFilters(old)
60          }
61          if err == nil {
62                  mapFilters, err = getMapFilters(old)
63          }
64          allItems := roaring.NewBitmap()
65          if err == nil && len(old.Versions) > 0 {
66                  _, err = allItems.FromBuffer(old.Versions[0].AllItems)
67          }
68          var cache *exactCache
69          if err == nil {
70                  cache, err = newExactCache(len(old.Versions))
71          }
72          if err == nil {
73                  items := roaring.NewBitmap()
74                  ctx := &exactContext{
75                          cache:   cache,
76                          config:  make(map[string]string),
77                          version: version,
78                  }
79                  hasher := eIndex.options.Hasher()
80                  saver = &exactBatch{
81                          indexer:      eIndex,
82                          ctx:          ctx,
83                          bitmapHasher: hasher,
84                          sources:      eIndex.getSources(cache, old),
85                          util:         getExactBatchUtil(),
86                          inserter:     eIndex.controller.Insert(ctx),
87                          old:          old,
88                          items:        items,
89                          allItems:     allItems,
90                          dataFilters:  dataFilters,
```

```
                        mapFilters:   mapFilters,
                }
        }
        return saver, err
}

func (eIndex *exactIndexer) newExactOptimizer(old exactMetaVersions, ctx
↪  *exactContext) (Optimizer, error) {
        var dataFiltersList, mapFiltersList []*bloom.BloomFilter
        err := globalErr
        if err == nil {
                dataFiltersList, err = getDataFilters(old)
        }
        if err == nil {
                mapFiltersList, err = getMapFilters(old)
        }
        var dataCount, mapCount uint
        dataFilters := make(map[string]*bloom.BloomFilter,
        ↪  len(dataFiltersList))
        for index, filter := range dataFiltersList {
                version := old.Versions[index]
                dataCount += uint(version.DataKeysCount)
                dataFilters[version.Version] = filter
        }
        mapFilters := make(map[string]*bloom.BloomFilter,
        ↪  len(mapFiltersList))
        for index, filter := range mapFiltersList {
                version := old.Versions[index]
                mapCount += uint(version.MapKeysCount)
                mapFilters[version.Version] = filter
        }
        var allItems *roaring.Bitmap
        if err == nil {
                allItems = roaring.New()
                _, err = allItems.FromBuffer(old.Versions[0].AllItems)
        }
        var optimizer *exactOptimizer
        if err == nil {
                optimizer = &exactOptimizer{
```

```
127                         dataFilters: dataFilters,
128                         mapFilters:  mapFilters,
129                         indexer:     eIndex,
130                         old:         old,
131                         dataMeta:    bloom.NewWithEstimates(dataCount,
                            ↪  eIndex.options.FalsePositiveKeys),
132                         keys:        bloom.NewWithEstimates(mapCount,
                            ↪  eIndex.options.FalsePositiveKeys),
133                         allItems:    allItems,
134                         ctx:         ctx,
135                 }
136         }
137         return optimizer, err
138 }
139
140 func (eIndex *exactIndexer) getSources(cache *exactCache, old
    ↪  exactMetaVersions) []ContextGetter {
141         sources := make([]ContextGetter, len(old.Versions))
142         for index, version := range old.Versions {
143                 sources[index] = old.GetContext(cache, version.Version)
144         }
145         return sources
146 }
147
148 func (eIndex *exactIndexer) Optimize() error {
149         var old exactMetaVersions
150         err := eIndex.storage.Get(eIndex.getMetaTable(),
        ↪  eIndex.getMetaKey(), &old)
151         if kv.IsKeyNotFoundError(err) {
152                 err = nil
153         }
154         if err == nil && len(old.Versions) >=
        ↪  eIndex.options.MergeMinVersions {
155                 var optimizer Optimizer
156                 var version string
157                 if err == nil {
158                         version, err = eIndex.generateNewVersionID()
159                 }
160                 var cache *exactCache
```

```
161                 if err == nil {
162                         cache, err = newExactCache(len(old.Versions))
163                 }
164                 target := &exactContext{
165                         config:  make(map[string]string),
166                         cache:   cache,
167                         version: version,
168                 }
169                 if err == nil {
170                         optimizer, err = eIndex.newExactOptimizer(old,
                        ↪  target)
171                 }
172                 if err == nil {
173                         sources := eIndex.getSources(cache, old)
174                         err = eIndex.controller.Optimize(sources,
                        ↪  optimizer, target)
175                 }
176                 if err == nil {
177                         err = eIndex.RunGC()
178                 }
179         }
180         return err
181 }
182 func (eIndex *exactIndexer) Delete() error {
183         metaTable := eIndex.getMetaTable()
184         err := globalErr
185         query := eIndex.storage.GetAllKeys(metaTable)
186         name := eIndex.options.Name + "-"
187         for err == nil {
188                 var key string
189                 var item exactMetaVersions
190                 key, err = query.Next(&item)
191                 if err == nil && strings.HasPrefix(key, name) {
192                         err = eIndex.storage.Del(metaTable, key)
193                 }
194         }
195         if kv.IsDoneError(err) {
196                 err = nil
197         }
```

```
198         if err == nil {
199                 err = eIndex.controller.Delete(&exactContext{version:
                    ↪  name + "-"}, false, []string{name + "-"})
200         }
201         return err
202 }
203
204 func (eIndex *exactIndexer) getResolver(version string) (exactResolver,
    ↪  error) {
205         var meta exactMetaVersions
206         if version == "" || version == "latest" {
207                 version = eIndex.options.Name + "-" + metaKey
208         }
209         err := globalErr
210         if err == nil {
211                 err = eIndex.storage.Get(eIndex.getMetaTable(), version,
                    ↪  &meta)
212         }
213         all := roaring.New()
214         if err == nil {
215                 _, err = all.FromBuffer(meta.Versions[0].AllItems)
216         }
217         if kv.IsKeyNotFoundError(err) {
218                 err = nil
219         }
220         var cache *exactCache
221         if err == nil {
222                 cache, err = newExactCache(len(meta.Versions))
223         }
224         var mapFilters []*bloom.BloomFilter
225         if err == nil {
226                 mapFilters, err = getMapFilters(meta)
227         }
228         var data exactResolver
229         if err == nil {
230                 data = exactResolver{
231                         all:        all,
232                         indexer:    eIndex,
233                         cache:      cache,
```

```
234                         meta:       meta,
235                         mapFilters: mapFilters,
236                 }
237         }
238         return data, err
239 }
240
241 func (eIndex *exactIndexer) getIterator(meta exactMetaVersions, result
    ↪ *roaring.Bitmap, options SearchOptions, info *IndexIteratorInfo)
    ↪ (roaring.IntIterable, error) {
242         var err error
243         if result == nil {
244                 result = emptyBitmap
245         }
246         info.Query.Count = result.GetCardinality()
247         info.Query.Version = "latest"
248         if len(meta.Versions) > 0 {
249                 info.Query.Version = meta.Versions[0].Version
250         }
251         if !result.IsEmpty() {
252                 info.Query.First = result.Minimum()
253                 info.Query.Last = result.Maximum()
254         }
255         if options.Before > 0 {
256                 // Remove all the values AFTER the before value
257                 result.RemoveRange(uint64(options.Before), 1<<32)
258                 if options.Limit > 0 && result.GetCardinality() >
                    ↪ uint64(options.Limit) {
259                         min := result.GetCardinality() -
                            ↪ uint64(options.Limit)
260                         var first uint32
261                         first, err = result.Select(uint32(min))
262                         if err == nil {
263                                 // And then remove the values BEFORE the
                                    ↪ window we want to see..
264                                 result.RemoveRange(0, uint64(first))
265                         }
266                 }
267         } else {
```

```
268                 result.RemoveRange(0, uint64(options.After)) // Remove
                    ↪  from start up to a value
269                 if options.Limit > 0 && result.GetCardinality() >
                    ↪  uint64(options.Limit) {
270                         var last uint32
271                         last, err = result.Select(options.Limit)
272                         if err == nil {
273                                 // Remove the records AFTER the window we
                                    ↪  want to see
274                                 result.RemoveRange(uint64(last), 1<<32)
275                         }
276                 }
277         }
278         if !result.IsEmpty() {
279                 info.Page.Before = result.Minimum()
280                 info.Page.After = result.Maximum() + 1
281         }
282         return result.Iterator(), err
283 }
284
285 func (eIndex *exactIndexer) Search(expression Expression) IndexIterator {
286         return eIndex.SearchWithOptions(expression, SearchOptions{})
287 }
288
289 func (eIndex *exactIndexer) SearchWithOptions(expression Expression,
    ↪  options SearchOptions) IndexIterator {
290         var it roaring.IntIterable
291         var info IndexIteratorInfo
292         var result *roaring.Bitmap
293         data, err := eIndex.getResolver(options.Version)
294         if err == nil {
295                 result, err = expression.EvaluateBitmap(data)
296         }
297         if err == nil {
298                 it, err = eIndex.getIterator(data.meta, result, options,
                    ↪  &info)
299         }
300         var items []*roaring.Bitmap
301         if err == nil {
```

```
302                 items, err = getItemsBitmaps(data.meta)
303         }
304         return &exactIterator{
305                 results: []roaring.IntIterable{it},
306                 cache:   data.cache,
307                 indexer: eIndex,
308                 err:     err,
309                 meta:    data.meta,
310                 items:   items,
311                 info:    info,
312         }
313 }
314
315 func (eIndex *exactIndexer) RunGC() error {
316         var meta exactMetaVersions
317         metaTable := eIndex.getMetaTable()
318         indexMetaKey := eIndex.getMetaKey()
319         name := eIndex.options.Name + "-"
320         err := eIndex.storage.Get(metaTable, indexMetaKey, &meta)
321         var it kv.Iterator
322         if err == nil {
323                 it = eIndex.storage.GetAllKeys(metaTable)
324         }
325         var keys []string
326         for err == nil {
327                 var item exactMetaVersions
328                 var key string
329                 key, err = it.Next(&item)
330                 if err != nil || !strings.HasPrefix(key, name) ||
                    ↪  meta.Includes(key) || key == indexMetaKey {
331                         continue
332                 }
333                 keys = append(keys, key)
334         }
335         if kv.IsDoneError(err) {
336                 err = nil
337         }
338         for _, key := range keys {
339                 if err == nil {
```

```
340                         err = eIndex.storage.Del(metaTable, key)
341                 }
342         }
343         // We only invoke the controller's delete method IF we have
            ↪  deleted some version key
344         // on the index. If, for some reason, the controller's delete
            ↪  method cannot be completed,
345         // as the index grows and new versions appear the index will be
            ↪  cleaned again completely,
346         // without problems.
347         if err == nil && len(keys) > 0 {
348                 versions := make([]string, len(meta.Versions))
349                 for i, version := range meta.Versions {
350                         versions[i] = version.Version
351                 }
352                 err = eIndex.controller.Delete(&exactContext{version:
                    ↪  name}, true, versions)
353         }
354         return err
355 }
356
357 // NewExactIndexer returns an indexer which just matches strings
    ↪  perfectly
358 func NewExactIndexer(storage kv.Store, controller Controller, options
    ↪  ExactOptions) Index {
359         options = getDefaultExactOptions(options)
360         return &exactIndexer{
361                 storage:    storage,
362                 options:    options,
363                 controller: controller,
364         }
365 }
```

Arquivo exact_iterator.go

```
1  package indexer
2
3  import (
4          "sync"
5
```

```
6           "github.com/RoaringBitmap/roaring"
7           "github.com/pkg/errors"
8   )
9
10  type exactIterator struct {
11          results []roaring.IntIterable
12          cache   *exactCache
13          indexer *exactIndexer
14          lock    sync.Mutex
15          cursor  uint32
16          err     error
17          meta    exactMetaVersions
18          items   []*roaring.Bitmap
19          info    IndexIteratorInfo
20  }
21
22  func (ei *exactIterator) Info() IndexIteratorInfo {
23          return ei.info
24  }
25
26  func (ei *exactIterator) Next(item interface{}) error {
27          ei.lock.Lock()
28          defer ei.lock.Unlock()
29          if ei.err == nil {
30                  for len(ei.results) > 0 && !ei.results[0].HasNext() {
31                          ei.results = ei.results[1:]
32                  }
33                  if len(ei.results) == 0 {
34                          return errDone
35                  }
36                  ei.cursor = ei.results[0].Next()
37                  var version string
38                  for i, data := range ei.items {
39                          if data.Contains(ei.cursor) {
40                                  version = ei.meta.Versions[i].Version
41                          }
42                  }
43                  if ei.err == nil {
44                          ctx := ei.meta.GetContext(ei.cache, version)
```

```
45                          ei.err = ei.indexer.controller.GetData(ctx,
                            ↪  ei.cursor, item)
46                          ei.err = errors.WithMessage(ei.err, "unexpected
                            ↪  error while iterating")
47                  }
48          }
49          return ei.err
50  }
```

Arquivo exact_models.go

```
1   package indexer
2
3   import "strings"
4
5   // exactMetaVersion should be ideally considered private: It is used just
    ↪  to
6   // host data related to an exactIndexer instance in the Store
7   type exactMetaVersion struct {
8           Version       string `datastore:"version,noindex"`
9           AllItems      []byte `datastore:"all_items,noindex"`      //
            ↪  Bitmap
10          Items         []byte `datastore:"items,noindex"`          //
            ↪  Bitmap
11          AllMapKeys    []byte `datastore:"all_map_keys,noindex"`   //
            ↪  Bloom Filter
12          AllDataKeys   []byte `datastore:"all_data_keys,noindex"`  //
            ↪  Bloom Filter
13          MapKeysCount  int    `datastore:"map_keys_count,noindex"` //
            ↪  Count
14          DataKeysCount int    `datastore:"map_keys_count,noindex"` //
            ↪  Count
15          Max           int32  `datastore:"max,noindex"`
16  }
17
18  type exactMetaConfig struct {
19          Version string `datastore:"version,noindex"`
20          Key     string `datastore:"key,noindex"`
21          Value   string `datastore:"value,noindex"`
22  }
```

```
23
24  // exactMetaVersions should be ideally considered private: It is used
    ↪  just to
25  // host data related to an exactIndexer instance in the Store
26  type exactMetaVersions struct {
27          Versions []exactMetaVersion `datastore:"versions,noindex"`
28          Config   []exactMetaConfig  `datastore:"config,noindex"`
29  }
30
31  func (emv exactMetaVersions) Includes(key string) (result bool) {
32          return emv.Find(key) > -1
33  }
34  func (emv exactMetaVersions) Find(key string) (result int) {
35          result = -1
36          for i, version := range emv.Versions {
37                  if strings.HasPrefix(key, version.Version) {
38                          result = i
39                          break
40                  }
41          }
42          return
43  }
44
45  func (emv exactMetaVersions) GetContext(cache *exactCache, version
    ↪  string) ContextGetter {
46          configLen := 0
47          for _, item := range emv.Config {
48                  if item.Version == version {
49                          configLen++
50                  }
51          }
52          ctx := &exactContext{
53                  version: version,
54                  cache:   cache,
55                  config:  make(map[string]string, configLen),
56          }
57          for _, item := range emv.Config {
58                  if item.Version == version {
59                          ctx.config[item.Key] = item.Value
```

```
60                  }
61          }
62          return ctx
63  }
```

Arquivo exact_optimizer.go

```
1   package indexer
2
3   import (
4           "github.com/RoaringBitmap/roaring"
5           "github.com/fjorgemota/gurudamatricula/util/kv"
6           "github.com/fjorgemota/gurudamatricula/util/pool"
7           "github.com/willf/bloom"
8   )
9
10  type exactOptimizer struct {
11          allItems    *roaring.Bitmap
12          dataFilters map[string]*bloom.BloomFilter
13          mapFilters  map[string]*bloom.BloomFilter
14          keys        *bloom.BloomFilter
15          dataMeta    *bloom.BloomFilter
16          old         exactMetaVersions
17          tmp         []byte
18          ctx         *exactContext
19          indexer     *exactIndexer
20  }
21
22  func (eo *exactOptimizer) HasMapKey(ctx ContextGetter, key MapKey) bool {
23          eo.tmp = key.Bytes(eo.tmp)
24          version := ctx.GetVersion()
25          result := eo.mapFilters[version].Test(eo.tmp)
26          eo.tmp = eo.tmp[:0]
27          return result
28  }
29
30  func (eo *exactOptimizer) HasDataMeta(ctx ContextGetter, key string) bool
    ↪ {
31          eo.tmp = append(eo.tmp, key...)
32          version := ctx.GetVersion()
```

```
33          result := eo.dataFilters[version].Test(eo.tmp)
34          eo.tmp = eo.tmp[:0]
35          return result
36  }
37
38  func (eo *exactOptimizer) IsDeleted(counter uint32) bool {
39          return !eo.allItems.Contains(counter)
40  }
41
42  func (eo *exactOptimizer) AddDataMeta(key string) error {
43          eo.tmp = append(eo.tmp, key...)
44          eo.dataMeta.Add(eo.tmp)
45          eo.tmp = eo.tmp[:0]
46          return nil
47
48  }
49  func (eo *exactOptimizer) AddMapKey(key MapKey) error {
50          eo.tmp = key.Bytes(eo.tmp)
51          eo.keys.Add(eo.tmp)
52          eo.tmp = eo.tmp[:0]
53          return nil
54  }
55  func (eo *exactOptimizer) RemoveDeletedKeys(bitmap *roaring.Bitmap)
    ↪  (*roaring.Bitmap, error) {
56          bitmap.And(eo.allItems)
57          return bitmap, nil
58  }
59
60  func (eo *exactOptimizer) Commit() error {
61          tmpBuf := pool.GetBytesBuffer()
62          defer pool.PutBytesBuffer(tmpBuf)
63          eo.allItems.RunOptimize()
64          _, err := eo.allItems.WriteTo(tmpBuf)
65          encodedAllItems := tmpBuf.Len()
66          var encodedMapKeys int
67          if err == nil {
68                  _, err = eo.keys.WriteTo(tmpBuf)
69                  encodedMapKeys = tmpBuf.Len()
70          }
```

```
71          var encodedDataKeys int
72          if err == nil {
73                  _, err = eo.dataMeta.WriteTo(tmpBuf)
74                  encodedDataKeys = tmpBuf.Len()
75          }
76          if err == nil {
77                  // If all went OK, we can update the meta table to have
                    ↪  the right version
78                  bs := tmpBuf.Bytes()
79                  metaTable := eo.indexer.getMetaTable()
80                  indexMetaKey := eo.indexer.getMetaKey()
81                  err =
                    ↪  eo.indexer.storage.Transaction([]kv.Reference{{Table:
                    ↪  metaTable, Key: indexMetaKey}, {Table: metaTable, Key:
                    ↪  eo.ctx.GetVersion()}}, func(db kv.LimitedStore) error
                    ↪  {
82                          var oldMeta, newMeta exactMetaVersions
83                          var metaVersion exactMetaVersion
84                          errOp := db.Get(metaTable, indexMetaKey,
                            ↪  &oldMeta)
85                          if errOp == nil {
86                                  errOp = errConflict
87                                  versions1 := oldMeta.Versions
88                                  versions2 := eo.old.Versions
89                                  if len(versions1) == len(versions2) {
90                                          count := 0
91                                          for i := range versions2 {
92                                                  if versions1[i].Version
                                                    ↪  ==
                                                    ↪  versions2[i].Version
                                                    ↪  {
93                                                          count++
94                                                  }
95                                          }
96                                          if count == len(versions2) {
97                                                  errOp = nil
98                                          }
99                                  }
100                         }
```

```
                        metaVersion = exactMetaVersion{
                                Version:     eo.ctx.GetVersion(),
                                AllItems:    bs[:encodedAllItems],
                                Items:       bs[:encodedAllItems],
                                AllMapKeys:
                                ↪  bs[encodedAllItems:encodedMapKeys],
                                AllDataKeys:
                                ↪  bs[encodedMapKeys:encodedDataKeys],
                                Max:         eo.old.Versions[0].Max,
                        }
                        newMeta.Versions = append(newMeta.Versions,
                        ↪  metaVersion)
                        for key, value := range eo.ctx.config {
                                newMeta.Config = append(newMeta.Config,
                                ↪  exactMetaConfig{
                                        Version: eo.ctx.GetVersion(),
                                        Key:     key,
                                        Value:   value,
                                })
                        }
                        if errOp == nil {
                                errOp = db.Set(eo.indexer.getMetaTable(),
                                ↪  eo.ctx.GetVersion(), &newMeta)
                        }
                        if errOp == nil {
                                errOp = db.Set(eo.indexer.getMetaTable(),
                                ↪  indexMetaKey, &newMeta)
                        }
                        return errOp
                })
        }
        return err
}
```

Arquivo exact_options.go

```
package indexer

import (
        "hash"
```

```
        "hash/fnv"
)

// ExactOptions define a few options that are customizable on
↪  ExactIndexer
type ExactOptions struct {
        Name                 string
        DefaultField         string
        MetaTable            string
        FalsePositiveKeys    float64
        MergeMinVersions     int
        AutoMergeMinVersions int
        Hasher               func() hash.Hash
}

func getDefaultExactOptions(options ExactOptions) ExactOptions {
        if options.DefaultField == "" {
                options.DefaultField = "default"
        }
        if options.MetaTable == "" {
                options.MetaTable = options.Name + "-meta"
        }
        if options.FalsePositiveKeys <= 0 {
                options.FalsePositiveKeys = 0.01
        }
        if options.MergeMinVersions < 2 {
                options.MergeMinVersions = 2
        }
        if options.Hasher == nil {
                options.Hasher = fnv.New128a
        }
        return options
}
```

Arquivo exact_pool.go

```
package indexer

import "sync"

```

```
type exactBatchUtil struct {
	dataMeta     map[string]InputDataMeta
	bitmaps      map[string]Bitmap
	oldItems     map[string]struct{}
	stringIntern map[string]string
	keys         map[MapKey]Map
	tmp          []byte
}

var exactBatchUtilPool = sync.Pool{
	New: func() interface{} {
		return &exactBatchUtil{
			dataMeta:     make(map[string]InputDataMeta),
			bitmaps:      make(map[string]Bitmap),
			oldItems:     make(map[string]struct{}),
			stringIntern: make(map[string]string),
			keys:         make(map[MapKey]Map),
		}
	},
}

func getExactBatchUtil() *exactBatchUtil {
	return exactBatchUtilPool.Get().(*exactBatchUtil)
}

func putExactBatchUtil(value *exactBatchUtil) {
	for key := range value.dataMeta {
		delete(value.dataMeta, key)
	}
	for key := range value.bitmaps {
		delete(value.bitmaps, key)
	}
	for key := range value.oldItems {
		delete(value.oldItems, key)
	}
	for key := range value.stringIntern {
		delete(value.stringIntern, key)
	}
	for key := range value.keys {
```

```
                delete(value.keys, key)
        }
        value.tmp = value.tmp[:0]
        exactBatchUtilPool.Put(value)
}
```

Arquivo exact_resolver.go

```
package indexer

import (
        "github.com/RoaringBitmap/roaring"
        "github.com/willf/bloom"
)

type exactResolver struct {
        all        *roaring.Bitmap
        mapFilters []*bloom.BloomFilter
        meta       exactMetaVersions
        indexer    *exactIndexer
        cache      *exactCache
}

func (ea exactResolver) Get(value string) (*roaring.Bitmap, error) {
        return ea.GetField(ea.indexer.options.DefaultField, value)
}

func (ea exactResolver) GetField(field, value string) (*roaring.Bitmap,
↪  error) {
        var valueBitmap *roaring.Bitmap
        var err error
        cacheKey := "cache|" + field + "|" + value
        cached, ok := ea.cache.Get(cacheKey)
        if ok {
                valueBitmap, ok = cached.(*roaring.Bitmap)
        }
        if ok {
                return valueBitmap, err
        }
        mapKey := MapKey{Field: field, Value: value}
```

```
        bs := mapKey.Bytes(nil)
        bitmap := roaring.New()
        for i, version := range ea.meta.Versions {
                if err == nil && ea.mapFilters[i].Test(bs) {
                        cfg := ea.meta.GetContext(ea.cache,
                        ↪  version.Version)
                        err = ea.indexer.controller.GetMap(cfg, mapKey,
                        ↪  bitmap)
                        if err == nil {
                                if valueBitmap == nil {
                                        valueBitmap = bitmap
                                        bitmap = roaring.New()
                                } else {
                                        valueBitmap.Or(bitmap)
                                }
                        } else if IsNotFoundError(err) {
                                err = nil
                        }
                }
        }
        if valueBitmap != nil {
                valueBitmap.And(ea.all)
                ea.cache.Set(cacheKey, valueBitmap.Clone())
        }
        return valueBitmap, err
}

func (ea exactResolver) GetAll() (*roaring.Bitmap, error) {
        return ea.all.Clone(), nil
}
```

Arquivo expression.go

```
package indexer

import (
        "github.com/RoaringBitmap/roaring"
)

// All returns all results from the index
```

```
8  type All struct{}
9
10 // EvaluateBitmap returns the bitmap with the result IDs
11 func (op All) EvaluateBitmap(data ExpressionData) (*roaring.Bitmap,
   ↪  error) {
12         return data.GetAll()
13 }
14
15 // And returns the intersection between the results of the expressions
16 type And struct {
17         Operations []Expression
18 }
19
20 // EvaluateBitmap returns the bitmap with the result IDs
21 func (op And) EvaluateBitmap(data ExpressionData) (*roaring.Bitmap,
   ↪  error) {
22         var err error
23         var result *roaring.Bitmap
24         for _, operation := range op.Operations {
25                 var bitmap *roaring.Bitmap
26                 if err == nil {
27                         bitmap, err = operation.EvaluateBitmap(data)
28                 }
29                 if err == nil {
30                         if bitmap == nil {
31                                 bitmap = result
32                         }
33                         if result == nil {
34                                 result = bitmap
35                         } else {
36                                 result.And(bitmap)
37                         }
38                 }
39                 if result == nil || result.IsEmpty() {
40                         // Short-Circuit to stop loop when there were no
                           ↪  results already
41                         break
42                 }
43         }
```

```
44          return result, err
45  }
46
47  // NewAnd returns a new AND expression
48  func NewAnd(expressions ...Expression) Expression {
49          return And{expressions}
50  }
51
52  // Or returns the union between the results of the expressions
53  type Or struct {
54          Operations []Expression
55  }
56
57  // EvaluateBitmap returns the bitmap with the result IDs
58  func (op Or) EvaluateBitmap(data ExpressionData) (*roaring.Bitmap, error)
    ↪ {
59          var err error
60          var result *roaring.Bitmap
61          for _, operation := range op.Operations {
62                  var bitmap *roaring.Bitmap
63                  if err == nil {
64                          bitmap, err = operation.EvaluateBitmap(data)
65                  }
66                  if err == nil {
67                          if bitmap == nil {
68                                  bitmap = result
69                          }
70                          if result == nil {
71                                  result = bitmap
72                          } else {
73                                  result.Or(bitmap)
74                          }
75                  }
76          }
77          return result, err
78  }
79
80  // NewOr returns a new OR expression
81  func NewOr(expressions ...Expression) Expression {
```

```
82          return Or{expressions}
83  }
84
85  // Not returns all results that are not accepted by the Expression
86  type Not struct {
87          Operation Expression
88  }
89
90  // EvaluateBitmap returns the bitmap with the result IDs
91  func (op Not) EvaluateBitmap(data ExpressionData) (*roaring.Bitmap,
    ↪  error) {
92          var err error
93          var result, bitmap *roaring.Bitmap
94          if err == nil {
95                  result, err = data.GetAll()
96          }
97          if err == nil {
98                  bitmap, err = op.Operation.EvaluateBitmap(data)
99          }
100         if err == nil && bitmap != nil {
101                 result.AndNot(bitmap)
102         }
103         return result, err
104 }
105
106 // Terminal returns all results that match the specified terminal with
    ↪  default
107 // field
108 type Terminal struct {
109         Value string
110 }
111
112 // EvaluateBitmap returns the bitmap with the result IDs
113 func (op Terminal) EvaluateBitmap(data ExpressionData) (*roaring.Bitmap,
    ↪  error) {
114         return data.Get(op.Value)
115 }
116
```

```
117 // FieldTerminal returns all results that match the specified terminal
    ↪  with
118 // specified field
119 type FieldTerminal struct {
120         Attribute string
121         Value     string
122 }
123
124 // EvaluateBitmap returns the bitmap with the result IDs
125 func (op FieldTerminal) EvaluateBitmap(data ExpressionData)
    ↪  (*roaring.Bitmap, error) {
126         return data.GetField(op.Attribute, op.Value)
127 }
```

Arquivo fulltext.go

```
1  package indexer
2
3  import (
4          "github.com/RoaringBitmap/roaring"
5          "github.com/fjorgemota/gurudamatricula/util/kv"
6  )
7
8  const ftWordsTreeName = "words-tree"
9  const ftOriginalField = "original"
10 const ftExactField = "exact"
11
12 type ftFieldCache struct {
13         Type  string
14         Field string
15 }
16 type ftIndexer struct {
17         baseIndexer *exactIndexer
18         options     FullTextOptions
19         fieldCache  map[ftFieldCache]string
20 }
21
22 func (ft *ftIndexer) getFieldName(fieldType, field string) string {
23         var result string
24         var ok bool
```

```
25          term := ftFieldCache{
26                  Type:  fieldType,
27                  Field: field,
28          }
29          if result, ok = ft.fieldCache[term]; !ok {
30                  result = "_" + fieldType + ":" + field
31                  ft.fieldCache[term] = result
32          }
33          return result
34  }
35
36  func (ft *ftIndexer) newFullBatch(old exactMetaVersions) (Batch, error) {
37          exactBatch, err := ft.baseIndexer.newExactBatch(old)
38          var batch Batch
39          if err == nil {
40                  batch = &ftBatch{
41                          saver:   exactBatch,
42                          indexer: ft,
43                  }
44          }
45          return batch, err
46  }
47
48  func (ft *ftIndexer) Rebuild() (Batch, error) {
49          return ft.newFullBatch(exactMetaVersions{})
50  }
51
52  func (ft *ftIndexer) Update() (Batch, error) {
53          var old exactMetaVersions
54          err := ft.baseIndexer.storage.Get(ft.baseIndexer.getMetaTable(),
          ↪  ft.baseIndexer.getMetaKey(), &old)
55          if kv.IsKeyNotFoundError(err) {
56                  return ft.Rebuild()
57          }
58          var batch Batch
59          if err == nil {
60                  batch, err = ft.newFullBatch(old)
61          }
62          return batch, err
```

```
63 }
64
65 func (ft *ftIndexer) newFullOptimizer(old exactMetaVersions, cache
   ↪  *exactCache, sources []ContextGetter, target *exactContext)
   ↪  (Optimizer, error) {
66         optimizer, err := ft.baseIndexer.newExactOptimizer(old, target)
67         var result Optimizer
68         if err == nil {
69                 result = &ftOptimizer{
70                         cache:      cache,
71                         base:       optimizer,
72                         sources:    sources,
73                         controller: ft.baseIndexer.controller,
74                         inserter:
                           ↪  ft.baseIndexer.controller.Insert(target),
75                 }
76         }
77         return result, err
78 }
79
80 func (ft *ftIndexer) Optimize() error {
81         var old exactMetaVersions
82         err := ft.baseIndexer.storage.Get(ft.baseIndexer.getMetaTable(),
           ↪  ft.baseIndexer.getMetaKey(), &old)
83         if kv.IsKeyNotFoundError(err) {
84                 err = nil
85         }
86         if err == nil && len(old.Versions) >=
           ↪  ft.baseIndexer.options.MergeMinVersions {
87                 var optimizer Optimizer
88                 var cache *exactCache
89                 if err == nil {
90                         cache, err = newExactCache(len(old.Versions))
91                 }
92                 sources := ft.baseIndexer.getSources(cache, old)
93                 var version string
94                 if err == nil {
95                         version, err =
                           ↪  ft.baseIndexer.generateNewVersionID()
```

```
96                  }
97                  target := &exactContext{
98                          cache:   cache,
99                          config:  make(map[string]string),
100                         version: version,
101                 }
102                 if err == nil {
103                         optimizer, err = ft.newFullOptimizer(old, cache,
                            ↪  sources, target)
104                 }
105                 if err == nil {
106                         err = ft.baseIndexer.controller.Optimize(sources,
                            ↪  optimizer, target)
107                 }
108                 if err == nil {
109                         err = ft.RunGC()
110                 }
111         }
112         return err
113 }
114
115 func (ft *ftIndexer) Search(expression Expression) IndexIterator {
116         return ft.SearchWithOptions(expression, SearchOptions{})
117 }
118
119 func (ft *ftIndexer) SearchWithOptions(expression Expression, options
    ↪  SearchOptions) IndexIterator {
120         var it roaring.IntIterable
121         var info IndexIteratorInfo
122         var result *roaring.Bitmap
123         exactData, err := ft.baseIndexer.getResolver(options.Version)
124         if err == nil {
125                 data := ftResolver{
126                         indexer:           ft,
127                         acessor:           exactData,
128                         enableCorrections: false,
129                 }
130                 result, err = expression.EvaluateBitmap(data)
```

```
131                 if err == nil && (result == nil || result.IsEmpty()) &&
                    ↪  ft.options.MaxDistance > 0 {
132                         data.enableCorrections = true
133                         result, err = expression.EvaluateBitmap(data)
134                 }
135         }
136         if err == nil {
137                 it, err = ft.baseIndexer.getIterator(exactData.meta,
                    ↪  result, options, &info)
138         }
139         var items []*roaring.Bitmap
140         if err == nil {
141                 items, err = getItemsBitmaps(exactData.meta)
142         }
143         return &exactIterator{
144                 results: []roaring.IntIterable{it},
145                 cache:   exactData.cache,
146                 indexer: ft.baseIndexer,
147                 err:     err,
148                 meta:    exactData.meta,
149                 items:   items,
150                 info:    info,
151         }
152 }
153
154 func (ind *ftIndexer) Delete() error {
155         return ind.baseIndexer.Delete()
156 }
157
158 func (ind *ftIndexer) RunGC() error {
159         return ind.baseIndexer.RunGC()
160 }
161
162 func NewFullTextIndexer(storage kv.Store, controller Controller,
    ↪  exactOptions ExactOptions, fullTextOptions FullTextOptions) Index {
163         fullTextOptions = getDefaultFullTextOptions(fullTextOptions)
164         exact := NewExactIndexer(storage, controller,
            ↪  exactOptions).(*exactIndexer)
165         return &ftIndexer{
```

```
                baseIndexer: exact,
                options:     fullTextOptions,
                fieldCache:  make(map[ftFieldCache]string),
        }
}
```

Arquivo fulltext_batch.go

```
package indexer

import (
        "sort"
        "strings"
        "sync"

        "github.com/RoaringBitmap/roaring"
        "github.com/fjorgemota/gurudamatricula/util/indexer/stringset"
)

type ftBitmapChar struct {
        field   string
        value   string
        bitmaps map[string]struct{}
}

type ftBatch struct {
        saver   *exactBatch
        indexer *ftIndexer
        lock    sync.Mutex
}

func (fts *ftBatch) addMapKey(char ftBitmapChar) error {
        var err error
        if len(char.field) == 0 || len(char.value) == 0 {
                return err
        }
        newKey := MapKey{
                Field: char.field,
                Value: char.value,
        }
```

```
33          mapKey, ok := fts.saver.util.keys[newKey]
34          var result *roaring.Bitmap
35          if ok {
36                  oldBitmap := fts.saver.util.bitmaps[mapKey.Bitmap]
37                  oldBitmap.Count--
38                  result = oldBitmap.Bitmap
39                  delete(fts.saver.util.bitmaps, mapKey.Bitmap)
40                  if oldBitmap.Count > 0 {
41                          result = oldBitmap.Bitmap.Clone()
42                          fts.saver.util.bitmaps[mapKey.Bitmap] = oldBitmap
43                  }
44          } else {
45                  mapKey.Index = uint32(len(fts.saver.util.keys))
46                  result = roaring.New()
47          }
48          for bitmapKey := range char.bitmaps {
49                  if bitmapKey != mapKey.Bitmap {
50                          result.Or(fts.saver.util.bitmaps[bitmapKey].Bitmap)
51                  }
52          }
53          var resultEncoded string
54          resultEncoded, err = fts.saver.bitmapEncode(result)
55          var bitmap Bitmap
56          bitmap, ok = fts.saver.util.bitmaps[resultEncoded]
57          if ok {
58                  result = bitmap.Bitmap
59          }
60          bitmap.Bitmap = result
61          bitmap.Count++
62          fts.saver.util.bitmaps[resultEncoded] = bitmap
63          mapKey.Bitmap = resultEncoded
64          fts.saver.util.keys[newKey] = mapKey
65          for key, value := range fts.saver.util.dataMeta {
66                  if result.Contains(value.Counter) &&
                    ↪  value.Keys.CheckedAdd(mapKey.Index) {
67                          fts.saver.util.dataMeta[key] = value
68                  }
69          }
70          return err
```

```
}

func (fts *ftBatch) Commit() error {
        fts.lock.Lock()
        defer fts.lock.Unlock()
        var err error
        original := fts.indexer.getFieldName(ftOriginalField, "")
        originalLen := len(original)
        var originalCount int
        for key := range fts.saver.util.keys {
                if strings.HasPrefix(key.Field, original) {
                        originalCount++
                }
        }
        keyList := make(mapKeySort, originalCount)
        var maxLen uint
        for key := range fts.saver.util.keys {
                if strings.HasPrefix(key.Field, original) {
                        originalCount--
                        if uint(len(key.Value)) > maxLen {
                                maxLen = uint(len(key.Value))
                        }
                        keyList[originalCount] = key
                }
        }
        var nodes []TreeNode
        if err == nil && len(keyList) > 0 {
                sort.Sort(keyList)
                tree := stringset.New()
                var seenValue string
                var seenNode *stringset.Node

                var zero struct{}
                if fts.indexer.options.MaxPrefixLength < maxLen {
                        maxLen = fts.indexer.options.MaxPrefixLength
                }
                chars := make([]ftBitmapChar, maxLen)
                for i := range chars {
                        chars[i] = ftBitmapChar{
```

```
110                                     bitmaps:
                                        ↪  mapPool.Get().(map[string]struct{}),
111                                 }
112                         }
113                         for _, key := range keyList {
114                                 if err == nil && (seenValue != key.Field ||
                                    ↪  seenNode == nil) {
115                                         seenValue = key.Field
116                                         seenNode, err = tree.InsertPrefix(nil,
                                            ↪  seenValue[originalLen:])
117                                         if err == nil {
118                                                 seenNode, err =
                                                    ↪  tree.InsertPrefix(seenNode,
                                                    ↪  "|")
119                                         }
120                                 }
121                                 if err == nil {
122                                         value := key.Value
123                                         bitmap := fts.saver.util.keys[key].Bitmap
124
125                                         for i := range value {
126                                                 if i >= len(chars) {
127                                                         break
128                                                 }
129                                                 char := chars[i]
130                                                 if err == nil && (char.field !=
                                                    ↪  key.Field || char.value !=
                                                    ↪  value[:i+1]) {
131                                                         err = fts.addMapKey(char)
132                                                         char.field = key.Field
133                                                         char.value = value[:i+1]
134                                                         char.bitmaps =
                                                            ↪  cleanUnique(char.bitmaps)
135                                                 }
136                                                 char.bitmaps[bitmap] = zero
137                                                 chars[i] = char
138                                         }
139                                         if err == nil {
```

```
140                                         _, err = tree.InsertWord(seenNode,
                                            ↪  value)
141                                 }
142                         }
143                 }
144                 tree.Finish()
145                 for _, char := range chars {
146                         if err == nil {
147                                 err = fts.addMapKey(char)
148                         }
149                         char.bitmaps = cleanUnique(char.bitmaps)
150                         mapPool.Put(char.bitmaps)
151                 }
152                 nodes = getNodes(tree)
153         }
154         if err == nil {
155                 err = fts.saver.commit(func() error {
156                         var localErr error
157                         if len(nodes) > 0 {
158                                 localErr =
                                    ↪  fts.saver.inserter.CommitTree(ftWordsTreeName,
                                    ↪  nodes)
159                         }
160                         return localErr
161                 })
162         }
163         numVersions := len(fts.saver.old.Versions) + 1
164         nodes = nil
165         fts.saver = nil
166         if err == nil {
167                 autoMerge :=
                 ↪  fts.indexer.baseIndexer.options.AutoMergeMinVersions
168                 if autoMerge > 0 && autoMerge > numVersions {
169                         err = fts.indexer.Optimize()
170                 }
171         }
172         return err
173 }
174
```

```
175 func (fts *ftBatch) Delete(itemKey string) error {
176         return fts.saver.Delete(itemKey)
177 }
178
179 func (fts *ftBatch) Update(values map[string]string, itemKey string, item
    ↪  interface{}) error {
180         err := fts.Delete(itemKey)
181         for field, value := range values {
182                 if err == nil {
183                         err = fts.SetField(field, value, itemKey, item)
184                 }
185         }
186         return err
187 }
188
189 func (fts *ftBatch) Set(value, itemKey string, item interface{}) error {
190         return fts.SetField(fts.saver.indexer.options.DefaultField, value,
          ↪  itemKey, item)
191 }
192
193 func (fts *ftBatch) SetField(field, value, itemKey string, item
    ↪  interface{}) error {
194         fts.lock.Lock()
195         defer fts.lock.Unlock()
196         if fts.indexer.options.IsExact(field) {
197                 exactField := fts.indexer.getFieldName(ftExactField,
                  ↪  field)
198                 return fts.saver.SetField(exactField, value, itemKey,
                  ↪  item)
199         }
200         originalField := fts.indexer.getFieldName(ftOriginalField, field)
201         tokens := fts.indexer.options.Tokenizer(field, value)
202         var err error
203         for _, token := range tokens {
204                 if err == nil {
205                         err = fts.saver.SetField(originalField, token,
                          ↪  itemKey, item)
206                 }
207         }
```

```
208         return err
209 }
```

Arquivo fulltext_optimizer.go

```
1  package indexer
2
3  import (
4          "container/heap"
5
6          "github.com/RoaringBitmap/roaring"
7          "github.com/fjorgemota/gurudamatricula/util/indexer/stringset"
8  )
9
10 type ftTreeOptimizerQueueItem struct {
11         node    uint32
12         source  uint16
13         fullBuf string
14         parent  *stringset.Node
15 }
16
17 type ftTreeOptimizerQueue []ftTreeOptimizerQueueItem
18
19 func (fttoq ftTreeOptimizerQueue) Len() int {
20         return len(fttoq)
21 }
22
23 func (fttoq ftTreeOptimizerQueue) Swap(i, j int) {
24         fttoq[i], fttoq[j] = fttoq[j], fttoq[i]
25 }
26
27 func (fttoq ftTreeOptimizerQueue) Less(i, j int) bool {
28         return fttoq[i].fullBuf < fttoq[j].fullBuf
29 }
30
31 func (fttoq *ftTreeOptimizerQueue) Push(value interface{}) {
32         *fttoq = append(*fttoq, value.(ftTreeOptimizerQueueItem))
33 }
34
35 func (fttoq *ftTreeOptimizerQueue) Pop() interface{} {
```

```
36          old := *fttoq
37          l := fttoq.Len() - 1
38          result := old[l]
39          *fttoq = old[:l]
40          return result
41  }
42
43  type ftOptimizer struct {
44          base       Optimizer
45          sources    []ContextGetter
46          controller Controller
47          inserter   Inserter
48          cache      *exactCache
49  }
50
51  func (fto *ftOptimizer) HasMapKey(ctx ContextGetter, key MapKey) bool {
52          return fto.base.HasMapKey(ctx, key)
53  }
54
55  func (fto *ftOptimizer) HasDataMeta(ctx ContextGetter, key string) bool {
56          return fto.base.HasDataMeta(ctx, key)
57  }
58
59  func (fto *ftOptimizer) IsDeleted(counter uint32) bool {
60          return fto.base.IsDeleted(counter)
61  }
62
63  func (fto *ftOptimizer) AddDataMeta(key string) error {
64          return fto.base.AddDataMeta(key)
65  }
66
67  func (fto *ftOptimizer) AddMapKey(key MapKey) error {
68          return fto.base.AddMapKey(key)
69  }
70
71  func (fto *ftOptimizer) RemoveDeletedKeys(bitmap *roaring.Bitmap)
    ↪  (*roaring.Bitmap, error) {
72          return fto.base.RemoveDeletedKeys(bitmap)
73  }
```

```
74
75  func (fto *ftOptimizer) commitTree() error {
76          var err error
77          queue := &ftTreeOptimizerQueue{}
78          for sourceID := range fto.sources {
79                  heap.Push(queue, ftTreeOptimizerQueueItem{
80                          source: uint16(sourceID),
81                          node:   0,
82                  })
83          }
84          treeBuilder := stringset.New()
85          var node TreeNode
86          for err == nil && queue.Len() > 0 {
87                  item := heap.Pop(queue).(ftTreeOptimizerQueueItem)
88                  err = fto.controller.GetTreeNode(fto.sources[item.source],
                    ↪  ftWordsTreeName, item.node, &node)
89                  if item.node == 0 && IsNotFoundError(err) {
90                          // It may happen that some versions don't have a
                            ↪  tree, because
91                          // trees are incremental with each other and some
                            ↪  versions may just
92                          // delete a bunch of things, so we need to ignore
                            ↪  not found errors
93                          // while trying to load the root of these trees
94                          err = nil
95                          continue
96                  }
97                  parent := item.parent
98                  bufLen := len(item.fullBuf)
99                  var key string
100                 if bufLen > 0 {
101                         key = item.fullBuf[bufLen-1:]
102                 }
103                 if err == nil {
104                         if node.Final {
105                                 parent, err =
                                    ↪  treeBuilder.InsertWord(parent, key)
106                         } else {
```

```
107                                 parent, err =
                                    ↪ treeBuilder.InsertPrefix(parent, key)
108                         }
109                 }
110                 if err == nil {
111                         for _, edge := range node.Edges {
112                                 heap.Push(queue,
                                    ↪ ftTreeOptimizerQueueItem{
113                                         source:  item.source,
114                                         node:    edge.Value,
115                                         fullBuf: item.fullBuf +
                                            ↪ string(edge.Key),
116                                         parent:  parent,
117                                 })
118                         }
119                 }
120         }
121         if err == nil {
122                 treeBuilder.Finish()
123                 nodes := getNodes(treeBuilder)
124                 err = fto.inserter.CommitTree(ftWordsTreeName, nodes)
125         }
126         return err
127 }
128
129 func (fto *ftOptimizer) Commit() error {
130         err := fto.commitTree()
131         if err == nil {
132                 err = fto.base.Commit()
133         }
134         return err
135 }
```

Arquivo fulltext_options.go

```
1  package indexer
2
3  import (
4          "regexp"
5          "strings"
```

```
6           "sync"
7           "unicode"
8
9           "golang.org/x/text/transform"
10          "golang.org/x/text/unicode/norm"
11  )
12
13  type FullTextOptions struct {
14          IsExact         func(field string) bool
15          Tokenizer       func(field, value string) []string
16          MaxUserDistance uint
17          MaxTreeDistance uint
18          MaxDistance     uint
19          MaxPrefixLength uint
20  }
21
22  func DefaultIsExact(_ string) bool {
23          return false
24  }
25
26  var removerPool = sync.Pool{
27          New: func() interface{} {
28                  return transform.Chain(norm.NFD,
                    ↪  transform.RemoveFunc(isMn), norm.NFC)
29          },
30  }
31
32  var removePontuaction = regexp.MustCompile("[^a-zA-Z0-9\\s]")
33
34  func isMn(r rune) bool {
35          return unicode.Is(unicode.Mn, r)
36  }
37
38  func DefaultTokenizer(field, value string) []string {
39          value = strings.ToLower(value)
40          remover := removerPool.Get().(transform.Transformer)
41          if clean, _, err := transform.String(remover, value); err == nil
            ↪ {
42                  value = clean
```

```
        }
        remover.Reset()
        removerPool.Put(remover)
        value = removePontuaction.ReplaceAllString(value, "")
        return strings.Fields(value)
}

func getDefaultFullTextOptions(options FullTextOptions) FullTextOptions {
        if options.IsExact == nil {
                options.IsExact = DefaultIsExact
        }
        if options.Tokenizer == nil {
                options.Tokenizer = DefaultTokenizer
        }
        return options
}
```

Arquivo fulltext_resolver.go

```
package indexer

import (
        "sort"

        "github.com/RoaringBitmap/roaring"
)

type ftTreePrefixQueueItem struct {
        node uint32
        // Full buffer of the edges navigated from the root to this node
        fullBuf string
}

type ftTreeSymSpellQueueItem struct {
        node uint32
        // Number of edges "ignored" (not part of the full buffer)
        ↪  already
        bufDistance uint32
        // Full buffer of the edges navigated from the root to this node
        fullBuf string
```

```
21          // Index of the next character to check on the user buffer
22          userIndex uint
23          // Number of characters "ignored" on the user buffer
24          userDistance uint
25  }
26
27  type ftResolver struct {
28          indexer           *ftIndexer
29          enableCorrections bool
30          acessor           exactResolver
31  }
32
33  func (fta ftResolver) GetAll() (*roaring.Bitmap, error) {
34          return fta.acessor.GetAll()
35  }
36
37  func (fta ftResolver) getEdge(edges []TreeEdge, value rune) (uint32,
    ↪  bool) {
38          edgeIndex := sort.Search(len(edges), func(i int) bool {
39                  return edges[i].Key >= value
40          })
41          var result uint32
42          if edgeIndex < len(edges) && edges[edgeIndex].Key == value {
43                  result = edges[edgeIndex].Value
44          }
45          // The root edge will never be referenced here, because the tree
            ↪  is acyclic.
46          return result, result > 0
47  }
48  func (fta ftResolver) getPrefixes(ctx ContextGetter, root TreeNode, value
    ↪  string, results map[string]struct{}) (map[string]struct{}, error) {
49          controller := fta.acessor.indexer.controller
50          var queue []ftTreePrefixQueueItem
51          queue = append(queue, ftTreePrefixQueueItem{
52                  node:    uint32(root.ID),
53                  fullBuf: value,
54          })
55          var err error
56          for len(queue) > 0 && err == nil {
```

```
57                  var elem ftTreePrefixQueueItem
58                  elem, queue = queue[0], queue[1:]
59                  var node TreeNode
60                  if elem.node == uint32(root.ID) {
61                          node = root
62                  } else {
63                          err = controller.GetTreeNode(ctx, ftWordsTreeName,
                            ↪  elem.node, &node)
64                  }
65                  if node.Final {
66                          results[elem.fullBuf] = zero
67                  }
68                  for _, edge := range node.Edges {
69                          nextBuf := elem.fullBuf + string(edge.Key)
70                          nextElem := ftTreePrefixQueueItem{
71                                  node:    edge.Value,
72                                  fullBuf: nextBuf,
73                          }
74                          queue = append(queue, nextElem)
75                  }
76          }
77          return results, err
78  }
79
80  func (fta ftResolver) symspell(ctx ContextGetter, root TreeNode, value
    ↪  []rune, results map[string]struct{}) (map[string]struct{}, error) {
81          controller := fta.acessor.indexer.controller
82          maxTreeDistance := fta.indexer.options.MaxTreeDistance
83          maxUserDistance := fta.indexer.options.MaxUserDistance
84          maxDistance := fta.indexer.options.MaxDistance
85          var queue []ftTreeSymSpellQueueItem
86          queue = append(queue, ftTreeSymSpellQueueItem{
87                  node: uint32(root.ID),
88          })
89          temp := make(map[ftTreeSymSpellQueueItem]struct{})
90          var err error
91          for len(queue) > 0 && err == nil {
92                  var elem ftTreeSymSpellQueueItem
93                  elem, queue = queue[0], queue[1:]
```

```
94                  var node TreeNode
95                  if elem.node == uint32(root.ID) {
96                          node = root
97                  } else {
98                          err = controller.GetTreeNode(ctx, ftWordsTreeName,
                            ↪  elem.node, &node)
99                  }
100                 if err == nil && node.Final && elem.userIndex ==
                    ↪  uint(len(value)) {
101                         results[elem.fullBuf] = zero
102                 }
103                 var actualChar rune
104                 // We need to check if userIndex is inside the string
105                 // If it is, we cannot use actualChar to compare with
                    ↪  edges and so on
106                 // because we're already in the end of the string
107                 compareChar := uint(len(value)) > elem.userIndex
108                 if compareChar {
109                         actualChar = value[elem.userIndex]
110                         if _, ok := fta.getEdge(node.Edges, actualChar);
                            ↪  !ok && elem.userDistance < maxUserDistance &&
                            ↪  (elem.userDistance+uint(elem.bufDistance)+1)
                            ↪  < maxDistance {
111                                 nextElem := ftTreeSymSpellQueueItem{
112                                         node:         elem.node,
113                                         fullBuf:      elem.fullBuf,
114                                         bufDistance:  elem.bufDistance,
115                                         userIndex:    elem.userIndex + 1,
116                                         userDistance: elem.userDistance +
                                            ↪  1,
117                                 }
118                                 if _, ok := temp[nextElem]; !ok {
119                                         temp[nextElem] = zero
120                                         queue = append(queue, nextElem)
121                                 }
122                         }
123                 }
124                 for _, edge := range node.Edges {
125                         nextBuf := elem.fullBuf + string(edge.Key)
```

```
126                         if compareChar && actualChar == edge.Key {
127                                 // Found an edge with that letter
128                                 // Consume character: increment userIndex
                                    ↪  without increasing userDistance
129                                 nextElem := ftTreeSymSpellQueueItem{
130                                         node:         edge.Value,
131                                         fullBuf:      nextBuf,
132                                         bufDistance:  elem.bufDistance,
133                                         userIndex:    elem.userIndex + 1,
134                                         userDistance: elem.userDistance,
135                                 }
136                                 if _, ok := temp[nextElem]; !ok {
137                                         temp[nextElem] = zero
138                                         queue = append(queue, nextElem)
139                                 }
140                         } else if
                            ↪  (elem.userDistance+uint(elem.bufDistance)+1)
                            ↪  < maxDistance && uint(elem.bufDistance) <
                            ↪  maxTreeDistance {
141                                 // Ignore edge
142                                 // Also doesn't change userIndex or
                                    ↪  userDistance
143                                 nextElem := ftTreeSymSpellQueueItem{
144                                         node:         edge.Value,
145                                         fullBuf:      nextBuf,
146                                         bufDistance:  elem.bufDistance +
                                            ↪  1,
147                                         userIndex:    elem.userIndex,
148                                         userDistance: elem.userDistance,
149                                 }
150                                 if _, ok := temp[nextElem]; !ok {
151                                         temp[nextElem] = zero
152                                         queue = append(queue, nextElem)
153                                 }
154                         }
155                 }
156                 delete(temp, elem)
157         }
158         return results, err
```

```
159 }
160
161 func (fta ftResolver) traverse(ctx ContextGetter, node *TreeNode, value
    ↪  []rune) (bool, error) {
162         controller := fta.acessor.indexer.controller
163         var err error
164         ok := true
165         index := 0
166         for ok && err == nil && index < len(value) {
167                 var nextID uint32
168                 nextID, ok = fta.getEdge(node.Edges, value[index])
169                 if ok {
170                         err = controller.GetTreeNode(ctx, ftWordsTreeName,
                            ↪  nextID, node)
171                         index++
172                 }
173         }
174         return ok, err
175 }
176
177 func (fta ftResolver) prefixesVersion(ctx ContextGetter, rootNode []rune,
    ↪  value string, result map[string]struct{}) (map[string]struct{},
    ↪  error) {
178         controller := fta.acessor.indexer.controller
179         var node TreeNode
180         err := controller.GetTreeNode(ctx, ftWordsTreeName, uint32(0),
            ↪  &node)
181         var ok bool
182         if err == nil {
183                 // traverse will return false if the field cannot be
                    ↪  found, so
184                 // at least if the field doesn't exist we will
                    ↪  short-circuit
185                 // here
186                 ok, err = fta.traverse(ctx, &node, rootNode)
187         }
188         if ok {
189                 result, err = fta.getPrefixes(ctx, node, value, result)
190         }
```

```
191         return result, err
192 }
193
194 func (fta ftResolver) prefixes(field, value string, uniqueify
    ↪  map[string]struct{}) (map[string]struct{}, error) {
195         var err error
196         // Fields have that suffix to split between fields and values:
197         fieldRunes := []rune(field + "|" + value)
198         // Scan every version present in the tree?
199         for _, version := range fta.acessor.meta.Versions {
200                 uniqueify, err =
                    ↪  fta.prefixesVersion(fta.acessor.meta.GetContext(fta.acessor.cac
                    ↪  version.Version), fieldRunes, value, uniqueify)
201         }
202         return uniqueify, err
203 }
204
205 func (fta ftResolver) correctorVersion(ctx ContextGetter, rootNode, value
    ↪  []rune, result map[string]struct{}) (map[string]struct{}, error) {
206         controller := fta.acessor.indexer.controller
207         var node TreeNode
208         err := controller.GetTreeNode(ctx, ftWordsTreeName, uint32(0),
            ↪  &node)
209         var ok bool
210         if err == nil {
211                 // traverse will return false if the field cannot be
                    ↪  found, so
212                 // at least if the field doesn't exist we willl
                    ↪  short-circuit
213                 // here
214                 ok, err = fta.traverse(ctx, &node, rootNode)
215         }
216         if ok {
217                 result, err = fta.symspell(ctx, node, value, result)
218         }
219         return result, err
220 }
221
```

```
222 func (fta ftResolver) corrector(field, value string, uniqueify
    ↪  map[string]struct{}) (map[string]struct{}, error) {
223         var err error
224         if fta.enableCorrections {
225                 // Fields have that suffix to split between fields and
                    ↪  values:
226                 fieldRunes := []rune(field + "|")
227                 valueRunes := []rune(value)
228                 // Scan every version present in the tree?
229                 for _, version := range fta.acessor.meta.Versions {
230                         uniqueify, err =
                            ↪  fta.correctorVersion(fta.acessor.meta.GetContext(fta.acessor
                            ↪  version.Version), fieldRunes, valueRunes,
                            ↪  uniqueify)
231                 }
232         }
233         return uniqueify, err
234 }
235
236 func (fta ftResolver) Get(value string) (*roaring.Bitmap, error) {
237         return fta.GetField(fta.acessor.indexer.options.DefaultField,
            ↪  value)
238 }
239
240 func (fta ftResolver) GetField(field, value string) (*roaring.Bitmap,
    ↪  error) {
241         if fta.indexer.options.IsExact(field) {
242                 exactField := fta.indexer.getFieldName(ftExactField,
                    ↪  field)
243                 return fta.acessor.GetField(exactField, value)
244         }
245         tokens := fta.indexer.options.Tokenizer(field, value)
246         var valueBitmap *roaring.Bitmap
247         var err error
248         cacheKey := "cache|" + field + "|" + value
249         cached, ok := fta.acessor.cache.Get(cacheKey)
250         if ok {
251                 valueBitmap, ok = cached.(*roaring.Bitmap)
252         }
```

```
253         if ok {
254                 return valueBitmap, err
255         }
256         uniqueify := mapPool.Get().(map[string]struct{})
257         originalField := fta.indexer.getFieldName(ftOriginalField, field)
258         for _, token := range tokens {
259                 var mapKeyBitmap *roaring.Bitmap
260                 if err == nil {
261                         mapKeyBitmap, err =
                        ↪  fta.acessor.GetField(originalField, token)
262                 }
263                 if err == nil && (uint(len(token)) >
                ↪  fta.indexer.options.MaxPrefixLength ||
                ↪  mapKeyBitmap.IsEmpty()) {
264                         uniqueify, err = fta.prefixes(field, token,
                        ↪  uniqueify)
265                         for prefix := range uniqueify {
266                                 var prefixBitmap *roaring.Bitmap
267                                 if err == nil {
268                                         prefixBitmap, err =
                                        ↪  fta.acessor.GetField(originalField,
                                        ↪  prefix)
269                                 }
270                                 if prefixBitmap != nil {
271                                         if mapKeyBitmap == nil {
272                                                 mapKeyBitmap =
                                                ↪  prefixBitmap
273                                         } else {
274                                                 mapKeyBitmap.Or(prefixBitmap)
275                                         }
276                                 }
277                         }
278                         uniqueify = cleanUnique(uniqueify)
279                 }
280                 if err == nil && fta.enableCorrections &&
                ↪  fta.indexer.options.MaxDistance > 0 && (mapKeyBitmap
                ↪  == nil || mapKeyBitmap.IsEmpty()) {
281                         // Apply corrections
```

```
282                         uniqueify, err = fta.corrector(field, token,
                            ↪  uniqueify)
283                         for correction := range uniqueify {
284                                 var correctionBitmap *roaring.Bitmap
285                                 if err == nil {
286                                         correctionBitmap, err =
                                            ↪  fta.acessor.GetField(originalField,
                                            ↪  correction)
287                                 }
288                                 if correctionBitmap != nil {
289                                         if mapKeyBitmap == nil {
290                                                 mapKeyBitmap =
                                                    ↪  correctionBitmap
291                                         } else {
292                                                 mapKeyBitmap.Or(correctionBitmap)
293                                         }
294                                 }
295                         }
296                         uniqueify = cleanUnique(uniqueify)
297                 }
298                 if mapKeyBitmap != nil {
299                         if valueBitmap == nil {
300                                 valueBitmap = mapKeyBitmap
301                         } else {
302                                 valueBitmap.And(mapKeyBitmap)
303                         }
304                 }
305         }
306         mapPool.Put(uniqueify)
307         if valueBitmap != nil {
308                 valueBitmap.And(fta.acessor.all)
309                 fta.acessor.cache.Set(cacheKey, valueBitmap.Clone())
310         }
311         return valueBitmap, err
312 }
```

Arquivo mapkey.go

```
1 package indexer
2
```

```
// MapKey is a struct that defines a map key in the system
type MapKey struct {
        Field string
        Value string
}

func (mk MapKey) Bytes(result []byte) []byte {
        result = append(result, mk.Field...)
        result = append(result, '|')
        result = append(result, mk.Value...)
        return result
}

type mapKeySort []MapKey

func (m mapKeySort) Len() int {
        return len(m)
}

func (m mapKeySort) Swap(i, j int) {
        m[i], m[j] = m[j], m[i]
}

func (m mapKeySort) Less(i, j int) bool {
        a, b := m[i], m[j]
        if a.Field != b.Field {
                return a.Field < b.Field
        }
        return a.Value < b.Value
}
```

Arquivo shard_controller.go

```
package indexer

import (
        "bytes"
        "container/heap"
        "io"
        "math"
```

```
8           "sort"
9           "strconv"
10          "strings"
11
12          "github.com/RoaringBitmap/roaring"
13          "github.com/fjorgemota/gurudamatricula/util/coder"
14          "github.com/fjorgemota/gurudamatricula/util/kv"
15          "github.com/fjorgemota/gurudamatricula/util/pool"
16          "github.com/pkg/errors"
17  )
18
19  var errShardOverflow = errors.New("indexer: Detected shard overflow")
20
21  const dataMetaShards = "data-meta-shards"
22  const mapShards = "map-shards"
23  const dataShards = "data-shards"
24  const treeShards = "tree-shards-"
25
26  type Sharder interface {
27          GetNode(string) (string, error)
28  }
29
30  type shardIndexMapKey struct {
31          Encoded string `datastore:"encoded,noindex" json:"encoded"`
32          Field   int    `datastore:"field,noindex" json:"field"`
33          Value   string `datastore:"value,noindex" json:"value"`
34          Start   int    `datastore:"start,noindex" json:"start"`
35          End     int    `datastore:"end,noindex" json:"end"`
36  }
37
38  // shardIndexMapItem is used just to host data related to an Indexer
    ↪  instance in the
39  // Store
40  type shardIndexMapItem struct {
41          Fields []string          `datastore:"fields,noindex"
            ↪  json:"fields"`
42          Keys   []shardIndexMapKey `datastore:"keys,noindex" json:"keys"`
43          Items  []byte            `datastore:"items,noindex"
            ↪  json:"items"`
```

```
}

type shardIndexTreeNode struct {
        ID            int32 `datastore:"id,noindex" json:"id"`
        Final         bool  `datastore:"final,noindex" json:"final"`
        ChildrenStart int   `datastore:"children_start,noindex"
        ↪  json:"children_start"`
        ChildrenEnd   int   `datastore:"children_end,noindex"
        ↪  json:"children_end"`
}

type shardIndexTreeEdge struct {
        Key   rune  `datastore:"key,noindex" json:"key"`
        Value int32 `datastore:"value,noindex" json:"value"`
}

type shardIndexTree struct {
        Nodes []shardIndexTreeNode `datastore:"nodes,noindex"
        ↪  json:"nodes"`
        Edges []shardIndexTreeEdge `datastore:"edges,noindex"
        ↪  json:"edges"`
}

type shardIndexData struct {
        Data []byte  `datastore:"data,noindex" json:"data"`
        Keys []int32 `datastore:"keys,noindex" json:"keys"`
}

type shardIndexDataMetaItem struct {
        Counter int32  `datastore:"counter,noindex" json:"counter"`
        Deleted bool   `datastore:"deleted,noindex" json:"deleted"`
        Name    string `datastore:"name,noindex" json:"name"`
        Start   int    `datastore:"start,noindex" json:"start"`
        End     int    `datastore:"end,noindex" json:"end"`
}

type shardRange struct {
        Start int
        End   int
```

```
79 }
80
81 type shardMap struct {
82         ShardID uint16
83         Index   uint16
84 }
85
86 // shardIndexDataMeta is used just to host data related to an Indexer
   ↪  instance in the
87 // Store
88 type shardIndexDataMeta struct {
89         Items   []shardIndexDataMetaItem `datastore:"items_keys,noindex"
           ↪  json:"items_keys"`
90         KeysIDs []byte                   `datastore:"keys_ids,noindex"
           ↪  json:"keys_ids"`
91         Keys    []MapKey                 `datastore:"keys,noindex"
           ↪  json:"keys"`
92 }
93
94 type ShardKVOptions struct {
95         TablePrefix string
96         MaximumSize int
97         Encoder     func(MapKey) string
98         IntEncoder  coder.IntEncoder
99 }
100
101 func getDefaultShardKVOptions(options ShardKVOptions) ShardKVOptions {
102         if options.Encoder == nil {
103                 options.Encoder = kvEncoder
104         }
105         if options.IntEncoder == nil {
106                 options.IntEncoder = defaultEncoder
107         }
108         return options
109 }
110
111 type shardKVTables struct {
112         mapTable      string
113         dataMetaTable string
```

```
        dataTable     string
        treeTable     string
}

type shardKVController struct {
        tables       shardKVTables
        storage      kv.Store
        dependencies ShardDependencies
        options      ShardKVOptions
}

type shardBuf struct {
        buf *bytes.Buffer
}

func (sb *shardBuf) Write(p []byte) (int, error) {
        return sb.buf.Write(p)
}

func (sb *shardBuf) Len() int {
        return sb.buf.Len()
}

func (sb *shardBuf) Bytes() []byte {
        return sb.buf.Bytes()
}

func (sb *shardBuf) Reset() {
        sb.buf.Reset()
}

func (sb *shardBuf) Truncate(n int) {
        sb.buf.Truncate(n)
}

func (sb *shardBuf) Close() error {
        return nil
}

```

```
func (kc shardKVController) getShardKey(ctx ContextGetter, shard string)
↪  string {
        return ctx.GetVersion() + "-" + shard
}

type shardOptimizerItem struct {
        Source uint16
        Shard  uint16
        Index  uint16
}

type shardOptimizerQueue struct {
        Items    []shardOptimizerItem
        Comparer func(i, j shardOptimizerItem) bool
        Length   int
}

func (soq *shardOptimizerQueue) Push(x interface{}) {
        soq.Items[soq.Length] = x.(shardOptimizerItem)
        soq.Length++
}
func (soq *shardOptimizerQueue) Pop() interface{} {
        soq.Length--
        return soq.Items[soq.Length]
}

func (soq *shardOptimizerQueue) Len() int {
        return int(soq.Length)
}

func (soq *shardOptimizerQueue) Less(i, j int) bool {
        itemI := soq.Items[i]
        itemJ := soq.Items[j]
        return soq.Comparer(itemI, itemJ)
}

func (soq *shardOptimizerQueue) Swap(i, j int) {
        soq.Items[i], soq.Items[j] = soq.Items[j], soq.Items[i]
}
```

```
191
192 func (kc shardKVController) optimizeMap(sources []ContextGetter, handler
    ↪ Optimizer, target ContextSetter, dataBuf, shardsBuf *bytes.Buffer)
    ↪ error {
193         shards := make([][]string, len(sources))
194         mapList := make([]shardIndexMapItem, len(sources))
195         queue := &shardOptimizerQueue{
196                 Items: make([]shardOptimizerItem, len(sources)),
197                 Comparer: func(itemI, itemJ shardOptimizerItem) bool {
198                         return
                        ↪ mapList[itemI.Source].Keys[itemI.Index].Encoded
                        ↪ <
                        ↪ mapList[itemJ.Source].Keys[itemJ.Index].Encoded
199                 },
200         }
201         var err error
202         for sourceID, source := range sources {
203                 var sourceShards []string
204                 if err == nil {
205                         sourceShards, err = getStringShards(source,
                        ↪ mapShards)
206                 }
207                 if err == nil {
208                         shards[sourceID] = make([]string,
                        ↪ (len(sourceShards)-1)/2)
209                         for i, shard := range sourceShards {
210                                 if i%2 == 1 {
211                                         shards[sourceID][(i-1)/2] = shard
212                                 }
213                         }
214                         if len(shards[sourceID]) > 0 {
215                                 key := kc.getShardKey(source,
                                ↪ shards[sourceID][0])
216                                 mapList[sourceID] =
                                ↪ cleanShardIndexMapItem(mapList[sourceID])
217                                 err = kc.storage.Get(kc.tables.mapTable,
                                ↪ key, &mapList[sourceID])
218                                 if err == nil {
```

```
219                                     mapList[sourceID].Keys =
                                        ↪ fromLevelOrderShardIndexMapKey(mapList[sourc
220                             }
221                             heap.Push(queue, shardOptimizerItem{
222                                     Source: uint16(sourceID),
223                                     Shard:  0,
224                                     Index:  0,
225                             })
226                     }
227             }
228     }
229     mapItemRange := make(map[string]shardRange)
230     mapFields := make(map[string]int)
231     var totalSize int
232     totalSize += 32 // Overhead per record
233     totalSize += 7  // Fields (name)
234     totalSize += 1  // Fields overhead
235     totalSize += 5  // Keys (name)
236     totalSize += 1  // Keys overhead
237     totalSize += 6  // Items (name)
238     totalSize += 1  // Items overhead
239     var lastKey, lastShardKey string
240     var mapResult shardIndexMapItem
241     var shardID int
242     resultBitmap := roaring.New()
243     newBitmap := roaring.New()
244     for err == nil && queue.Len() > 0 {
245             queueItem := heap.Pop(queue).(shardOptimizerItem)
246             mapItem := mapList[queueItem.Source]
247             item := mapItem.Keys[queueItem.Index]
248             if err == nil && lastKey != item.Encoded {
249                     lastKey = item.Encoded
250                     if err == nil && totalSize+dataBuf.Len() >=
                        ↪ kc.options.MaximumSize {
251                             lastShardKey =
                                ↪ mapResult.Keys[len(mapResult.Keys)-1].Encoded
252                             err = addMapShard(kc, shardOptions{
253                                     dataBuf:   dataBuf,
254                                     shardsBuf: shardsBuf,
```

```
255                                     shardID:   shardID,
256                                     ctx:       target,
257                             }, mapResult)
258                             mapResult.Items = mapResult.Items[:0]
259                             mapResult.Fields = mapResult.Fields[:0]
260                             mapResult.Keys = mapResult.Keys[:0]
261                             totalSize = 32 // Overhead per record
262                             totalSize += 7 // Fields (name)
263                             totalSize += 1 // Fields overhead
264                             totalSize += 5 // Keys (name)
265                             totalSize += 1 // Keys overhead
266                             totalSize += 6 // Items (name)
267                             totalSize += 1 // Items overhead
268                             for key := range mapItemRange {
269                                     delete(mapItemRange, key)
270                             }
271                             for key := range mapFields {
272                                     delete(mapFields, key)
273                             }
274                             dataBuf.Reset()
275                             shardID++
276                     }
277                     mapKey := MapKey{
278                             Field: mapItem.Fields[item.Field],
279                             Value: item.Value,
280                     }
281                     _, err =
                     ↪  resultBitmap.FromBuffer(mapItem.Items[item.Start:item.E
282                     for i, l := 0, queue.Len(); i < l && err == nil;
                     ↪  i++ {
283                             otherQueueItem := queue.Items[i]
284                             otherMapItem :=
                             ↪  mapList[otherQueueItem.Source]
285                             otherItem :=
                             ↪  otherMapItem.Keys[otherQueueItem.Index]
286                             if err == nil && otherItem.Encoded ==
                             ↪  item.Encoded {
287                                     _, err =
                                     ↪  newBitmap.FromBuffer(otherMapItem.Items
```

```
288                                     if err == nil {
289                                             resultBitmap.Or(newBitmap)
290                                     }
291                             }
292                     }
293                     if err == nil {
294                             resultBitmap, err =
                                ↪  handler.RemoveDeletedKeys(resultBitmap)
295                             if err == nil {
296                                     resultBitmap.RunOptimize()
297                             }
298                     }
299                     if err == nil && !resultBitmap.IsEmpty() {
300                             var itemRange shardRange
301                             var ok bool
302                             start := dataBuf.Len()
303                             _, err = resultBitmap.WriteTo(dataBuf)
304                             bs := dataBuf.Bytes()[start:]
305                             if itemRange, ok =
                                ↪  mapItemRange[string(bs)]; !ok {
306                                     itemRange = shardRange{
307                                             Start: start,
308                                             End:   dataBuf.Len(),
309                                     }
310                                     mapItemRange[string(bs)] =
                                        ↪  itemRange
311                             } else {
312                                     dataBuf.Truncate(start)
313                             }
314                             var fieldIndex int
315                             if fieldIndex, ok =
                                ↪  mapFields[mapKey.Field]; !ok {
316                                     totalSize += len(mapKey.Field) +
                                        ↪  1
317                                     mapResult.Fields =
                                        ↪  append(mapResult.Fields,
                                        ↪  mapKey.Field)
318                                     fieldIndex =
                                        ↪  len(mapResult.Fields) - 1
```

```
319                             mapFields[mapKey.Field] =
                                ↪ fieldIndex
320                     }
321                     totalSize += 8                      //
                        ↪ Encoded (name)
322                     totalSize += len(item.Encoded) + 1 //
                        ↪ Encoded (value)
323                     totalSize += 6                      //
                        ↪ Field (name)
324                     totalSize += 8                      //
                        ↪ Field (value)
325                     totalSize += 6                      //
                        ↪ Value (name)
326                     totalSize += 8                      //
                        ↪ Value (value)
327                     totalSize += 6                      //
                        ↪ Start (name)
328                     totalSize += 8                      //
                        ↪ Start (value)
329                     totalSize += 4                      // End
                        ↪ (name)
330                     totalSize += 8                      // End
                        ↪ (value)
331                     mapResult.Keys = append(mapResult.Keys,
                        ↪ shardIndexMapKey{
332                             Encoded: item.Encoded,
333                             Field:   fieldIndex,
334                             Value:   mapKey.Value,
335                             Start:   itemRange.Start,
336                             End:     itemRange.End,
337                     })
338                     if err == nil {
339                             err = handler.AddMapKey(mapKey)
340                     }
341                     if err == nil {
342                             err = errShardOverflow
343                             if shardID < math.MaxUint16 &&
                                ↪ (len(mapResult.Keys)-1) <
                                ↪ math.MaxUint16 {
```

```
344                                             err = nil
345                                     }
346                             }
347                     }
348             }
349             if err == nil {
350                     queueItem.Index++
351                     if queueItem.Index >= uint16(len(mapItem.Keys)) {
352                             // Need to load next shard!
353                             queueItem.Shard++
354                             sourceShards := shards[queueItem.Source]
355                             queueItem.Index = 0
356                             if queueItem.Shard >=
                                ↪  uint16(len(sourceShards)) {
357                                     // Okay, all shards were already
                                        ↪  readed! Ignore it
358                                     continue
359                             }
360                             if err == nil {
361                                     source :=
                                        ↪  sources[queueItem.Source]
362                                     key := kc.getShardKey(source,
                                        ↪  sourceShards[int(queueItem.Shard)])
363                                     mapList[queueItem.Source] =
                                        ↪  cleanShardIndexMapItem(mapList[queueItem.Sou
364                                     err =
                                        ↪  kc.storage.Get(kc.tables.mapTable,
                                        ↪  key,
                                        ↪  &mapList[queueItem.Source])
365                                     if err == nil {
366                                             mapList[queueItem.Source].Keys
                                                ↪  =
                                                ↪  fromLevelOrderShardIndexMapKey(mapLi
367                                     }
368                             }
369                     }
370                     heap.Push(queue, queueItem)
371             }
372     }
```

```
373         if err == nil && len(mapResult.Keys) > 0 {
374                 lastShardKey =
                    ↪  mapResult.Keys[len(mapResult.Keys)-1].Encoded
375                 err = addMapShard(kc, shardOptions{
376                         dataBuf:   dataBuf,
377                         shardsBuf: shardsBuf,
378                         shardID:   shardID,
379                         ctx:       target,
380                 }, mapResult)
381         }
382         if err == nil && lastShardKey != "" {
383                 err = shardsBuf.WriteByte('|')
384                 if err == nil {
385                         _, err = shardsBuf.WriteString(lastShardKey)
386                 }
387         }
388         target.Set(mapShards, shardsBuf.String())
389         dataBuf.Reset()
390         shardsBuf.Reset()
391         return err
392 }
393
394 func (kc shardKVController) optimizeDataMeta(sources []ContextGetter,
    ↪  handler Optimizer, target ContextSetter, dataBuf, shardsBuf
    ↪  *bytes.Buffer) error {
395         shards := make([][]string, len(sources))
396         dataMetaList := make([]shardIndexDataMeta, len(sources))
397         queue := &shardOptimizerQueue{
398                 Items: make([]shardOptimizerItem, len(sources)),
399                 Comparer: func(itemI, itemJ shardOptimizerItem) bool {
400                         listItemI :=
                            ↪  dataMetaList[itemI.Source].Items[itemI.Index]
401                         listItemJ :=
                            ↪  dataMetaList[itemJ.Source].Items[itemJ.Index]
402                         if listItemI.Name != listItemJ.Name {
403                                 return listItemI.Name < listItemJ.Name
404                         }
405                         return itemI.Source < itemJ.Source
406                 },
```

```
407         }
408         firstShard, err := kc.options.IntEncoder.Encode(0)
409         for sourceID, source := range sources {
410                 var sourceShards []string
411                 if err == nil {
412                         sourceShards, err = getStringShards(source,
                        ↪  dataMetaShards)
413                 }
414                 if err == nil {
415                         shards[sourceID] = make([]string,
                        ↪  (len(sourceShards)-1)/2)
416                         for i, shard := range sourceShards {
417                                 if i%2 == 1 {
418                                         shards[sourceID][(i-1)/2] = shard
419                                 }
420                         }
421                         if len(shards[sourceID]) > 0 {
422                                 key := kc.getShardKey(source, firstShard)
423                                 dataMetaList[sourceID] =
                                ↪  cleanShardIndexDataMeta(dataMetaList[sourceID])
424                                 err =
                                ↪  kc.storage.Get(kc.tables.dataMetaTable,
                                ↪  key, &dataMetaList[sourceID])
425                                 if err == nil {
426                                         dataMetaList[sourceID].Items =
                                        ↪  fromLevelOrderShardIndexDataMetaItem(dataMet
427                                 }
428                                 heap.Push(queue, shardOptimizerItem{
429                                         Source: uint16(sourceID),
430                                         Shard:  0,
431                                         Index:  0,
432                                 })
433                         }
434                 }
435         }
436         var dataMetaResult shardIndexDataMeta
437         var totalSize int
438         mapKeyIndexes := make(map[MapKey]uint32)
439         dataKeys := roaring.New()
```

```
440         var lastShardKey, lastKey string
441         totalSize += 32 // Overhead per record
442         totalSize += 6  // Items (name)
443         totalSize += 1  // Items overhead
444         totalSize += 8  // KeyIDs (name)
445         totalSize += 1  // KeyIDs overhead
446         totalSize += 5  // Keys (name)
447         totalSize += 1  // Keys (overhead)
448         // shardCount := make(map[uint16]uint16)
449         var shardID int
450         tmpBitmap := roaring.New()
451         for err == nil && queue.Len() > 0 {
452                 queueItem := heap.Pop(queue).(shardOptimizerItem)
453                 dataMetaItem := dataMetaList[queueItem.Source]
454                 item := dataMetaItem.Items[queueItem.Index]
455                 if err == nil && lastKey != item.Name {
456                         lastKey = item.Name
457                         var data DataMeta
458                         err = kc.saveDataMetaResult(dataMetaItem, item,
                        ↪  tmpBitmap, &data)
459                         // Save same data under new Key, if it's not
                        ↪  deleted, of course
460                         if err == nil && !data.Deleted &&
                        ↪  !handler.IsDeleted(data.Counter) {
461                                 if totalSize+dataBuf.Len() >=
                                ↪  kc.options.MaximumSize {
462                                         lastShardKey =
                                        ↪  dataMetaResult.Items[len(dataMetaResult
463                                         err = addDataMetaShard(kc,
                                        ↪  shardOptions{
464                                                 dataBuf:   dataBuf,
465                                                 shardsBuf: shardsBuf,
466                                                 shardID:   shardID,
467                                                 ctx:       target,
468                                         }, dataMetaResult)
469                                         dataMetaResult.Items =
                                        ↪  dataMetaResult.Items[:0]
470                                         dataMetaResult.KeysIDs =
                                        ↪  dataMetaResult.KeysIDs[:0]
```

```
471                             dataMetaResult.Keys =
                                ↪ dataMetaResult.Keys[:0]
472                             for key := range mapKeyIndexes {
473                                     delete(mapKeyIndexes,
                                        ↪ key)
474                             }
475                             totalSize = 32 // Overhead per
                                ↪ record
476                             totalSize += 6 // Items (name)
477                             totalSize += 1 // Items overhead
478                             totalSize += 8 // KeyIDs (name)
479                             totalSize += 1 // KeyIDs overhead
480                             totalSize += 5 // Keys (name)
481                             totalSize += 1 // Keys (overhead)
482                             dataBuf.Reset()
483                             shardID++
484                     }
485                     for _, mapKey := range data.Keys {
486                             var mapKeyIndex uint32
487                             var ok bool
488                             if mapKeyIndex, ok =
                                ↪ mapKeyIndexes[mapKey]; !ok {
489                                     dataMetaResult.Keys =
                                        ↪ append(dataMetaResult.Keys,
                                        ↪ mapKey)
490                                     mapKeyIndex =
                                        ↪ uint32(len(dataMetaResult.Keys)
                                        ↪ - 1)
491                                     mapKeyIndexes[mapKey] =
                                        ↪ mapKeyIndex
492                                     totalSize += 6
                                        ↪ // Field (name)
493                                     totalSize +=
                                        ↪ len(mapKey.Field) + 1
                                        ↪ // Field (value)
494                                     totalSize += 6
                                        ↪ // Value (name)
```

```
495                                         totalSize +=
                                            ↪  len(mapKey.Value) + 1
                                            ↪  // Value (value)
496                                 }
497                                 dataKeys.Add(mapKeyIndex)
498                         }
499                         if err == nil {
500                                 totalSize += 5
                                    ↪  // Name (name)
501                                 totalSize += len(item.Name) + 1
                                    ↪  // Name (value)
502                                 totalSize += 8
                                    ↪  // Counter (name)
503                                 totalSize += 8
                                    ↪  // Counter (value)
504                                 totalSize += 8
                                    ↪  // Deleted (name)
505                                 totalSize += 1
                                    ↪  // Deleted (value)
506                                 totalSize += 6
                                    ↪  // Start (name)
507                                 totalSize += 8
                                    ↪  // Start (value)
508                                 totalSize += 4
                                    ↪  // End (name)
509                                 totalSize += 8
                                    ↪  // End (value)
510                                 dataKeys.RunOptimize()
511                                 start := dataBuf.Len()
512                                 _, err =
                                    ↪  dataKeys.WriteTo(dataBuf)
513                                 if err == nil {
514                                         dataMetaResult.Items =
                                            ↪  append(dataMetaResult.Items,
                                            ↪  shardIndexDataMetaItem{
515                                                 Name:
                                                    ↪  item.Name,
516                                                 Counter:
                                                    ↪  int32(item.Counter),
```

```
517                                                     Deleted:
                                                        ↪ item.Deleted,
518                                                     Start:   start,
519                                                     End:
                                                        ↪ dataBuf.Len(),
520                                                 })
521                                                 err =
                                                    ↪ handler.AddDataMeta(item.Name)
522                                         }
523                                 }
524                                 dataKeys.Clear()
525                         }
526                 }
527                 if err == nil {
528                         queueItem.Index++
529                         if queueItem.Index >=
                            ↪ uint16(len(dataMetaItem.Items)) {
530                                 // Need to load next shard!
531                                 sourceShards := shards[queueItem.Source]
532                                 queueItem.Shard++
533                                 queueItem.Index = 0
534                                 if queueItem.Shard >=
                                    ↪ uint16(len(sourceShards)) {
535                                         // Okay, all shards were already
                                            ↪ readed! Ignore it
536                                         continue
537                                 }
538                                 if err == nil {
539                                         shard :=
                                            ↪ sourceShards[queueItem.Shard]
540                                         source :=
                                            ↪ sources[queueItem.Source]
541                                         key := kc.getShardKey(source,
                                            ↪ shard)
542                                         dataMetaList[queueItem.Source] =
                                            ↪ cleanShardIndexDataMeta(dataMetaList[queueIt
```

```
543                                             err =
                                                ↪  kc.storage.Get(kc.tables.dataMetaTable,
                                                ↪  key,
                                                ↪  &dataMetaList[queueItem.Source])
544                                             if err == nil {
545                                                     dataMetaList[queueItem.Source].Item
                                                        ↪  =
                                                        ↪  fromLevelOrderShardIndexDataMet
546                                             }
547                                     }
548                             }
549                             heap.Push(queue, queueItem)
550                     }
551             }
552             if err == nil && len(dataMetaResult.Items) > 0 {
553                     lastShardKey =
                        ↪  dataMetaResult.Items[len(dataMetaResult.Items)-1].Name
554                     err = addDataMetaShard(kc, shardOptions{
555                             dataBuf:   dataBuf,
556                             shardsBuf: shardsBuf,
557                             shardID:   shardID,
558                             ctx:       target,
559                     }, dataMetaResult)
560             }
561             if err == nil && shardsBuf.Len() > 0 {
562                     err = shardsBuf.WriteByte('|')
563                     if err == nil {
564                             _, err = shardsBuf.WriteString(lastShardKey)
565                     }
566             }
567             target.Set(dataMetaShards, shardsBuf.String())
568             dataBuf.Reset()
569             shardsBuf.Reset()
570             return err
571     }
572     func (kc shardKVController) optimizeData(sources []ContextGetter, handler
        ↪  Optimizer, target ContextSetter, shardsBuf *bytes.Buffer) error {
573             shards := make([]numberShards, len(sources))
574             var err error
```

```
575         for sourceID, source := range sources {
576                 var sourceShards numberShards
577                 if err == nil {
578                         sourceShards, err = getNumberShards(source,
                        ↪  dataShards)
579                 }
580                 if err == nil && len(sourceShards.encoded) > 0 {
581                         // Because we go from newer to older
582                         shards[len(sources)-1-sourceID] = sourceShards
583                 }
584         }
585         var lastShardKey uint32
586         var shardCount int
587         var data shardIndexData
588         for sourceID, sourceShards := range shards {
589                 // To get correct source:
590                 sourceID = len(sources) - sourceID - 1
591                 source := sources[sourceID]
592                 for _, shard := range sourceShards.encoded {
593                         oldKey := kc.getShardKey(source, shard)
594                         newKey := kc.getShardKey(target, shard)
595                         if err == nil {
596                                 err = kc.storage.Get(kc.tables.dataTable,
                                ↪  oldKey, &data)
597                         }
598                         if err == nil {
599                                 // We will only ignore shards IF ALL the
                                ↪  keys it contains
600                                 // are deleted.
601                                 // If that's not the case, we will copy
                                ↪  ALL the shard
602                                 // because it's hard to filter the
                                ↪  encoded data from the
603                                 // whole blob of data
604                                 shouldIgnore := true
605                                 for _, key := range data.Keys {
606                                         if
                                        ↪  !handler.IsDeleted(uint32(key))
                                        ↪  {
```

```
607                                             shouldIgnore = false
608                                             break
609                                     }
610                             }
611                             if !shouldIgnore {
612                                     lastShardKey =
                                        ↪ uint32(data.Keys[len(data.Keys)-1])
613                                     err =
                                        ↪ kc.storage.Set(kc.tables.dataTable,
                                        ↪ newKey, &data)
614                                     if err == nil {
615                                             if shardsBuf.Len() > 0 {
616                                                     err =
                                                        ↪ shardsBuf.WriteByte('|'
617                                             }
618                                             if err == nil {
619                                                     _, err =
                                                        ↪ shardsBuf.WriteString(s
620                                             }
621                                             if err == nil {
622                                                     err =
                                                        ↪ shardsBuf.WriteByte('|'
623                                             }
624                                             if err == nil {
625                                                     _, err =
                                                        ↪ shardsBuf.WriteString(s
626                                             }
627                                     }
628                                     shardCount++
629                             }
630                     }
631                     data = clearShardIndexData(data)
632             }
633     }
634     if shardsBuf.Len() > 0 && err == nil {
635             err = shardsBuf.WriteByte('|')
636             if err == nil {
637                     _, err =
                        ↪ shardsBuf.WriteString(strconv.Itoa(int(lastShardKey)))
```

```
638                 }
639         }
640         target.Set(dataShards, shardsBuf.String())
641         shardsBuf.Reset()
642         return err
643 }
644
645 func (kc shardKVController) Optimize(sources []ContextGetter, handler
    ↪  Optimizer, target ContextSetter) error {
646         shardsBuf := pool.GetBytesBuffer()
647         dataBuf := pool.GetBytesBuffer()
648         err := kc.optimizeMap(sources, handler, target, dataBuf,
            ↪  shardsBuf)
649         if err == nil {
650                 err = kc.optimizeDataMeta(sources, handler, target,
                    ↪  dataBuf, shardsBuf)
651         }
652         pool.PutBytesBuffer(dataBuf)
653         if err == nil {
654                 err = kc.optimizeData(sources, handler, target,
                    ↪  shardsBuf)
655         }
656         pool.PutBytesBuffer(shardsBuf)
657         if err == nil {
658                 err = handler.Commit()
659         }
660         return err
661 }
662
663 func (kc shardKVController) Insert(ctx ContextSetter) Inserter {
664         shardBufferBase := pool.GetBytesBuffer()
665         shardBuffer := &shardBuf{buf: shardBufferBase}
666         inserter := &shardKVInserter{
667                 controller:  kc,
668                 ctx:         ctx,
669                 dataBuf:     shardBuffer,
670                 shardsBuf:   pool.GetBytesBuffer(),
671                 dataEncoder:
                    ↪  kc.dependencies.GetStreamingEncoder(shardBuffer),
```

```
672         }
673         if kc.options.MaximumSize <= 0 {
674                 inserter.err = errors.New("indexer: Maximum size should
                    ↪ be bigger than zero")
675         }
676         return inserter
677 }
678
679 func (kc shardKVController) GetMap(ctx ContextGetter, mapKey MapKey,
    ↪ bitmap *roaring.Bitmap) error {
680         shards, err := getStringShards(ctx, mapShards)
681         key := kc.options.Encoder(mapKey)
682         if err == nil {
683                 err = errNotFound
684                 l := len(shards)
685                 if l > 0 && key >= shards[0] && key <= shards[l-1] {
686                         err = nil
687                 }
688         }
689         var shardKey string
690         if err == nil {
691                 shardIndex := (sort.Search((len(shards)-1)/2, func(i int)
                    ↪ bool {
692                         return shards[i*2] > key
693                 }) * 2) - 1
694                 shardKey = kc.getShardKey(ctx, shards[shardIndex])
695         }
696         var dbResult shardIndexMapItem
697         if err == nil {
698                 cacheKey := kc.tables.mapTable + "-" + shardKey
699                 tmp, ok := ctx.CacheGet(cacheKey)
700                 if ok {
701                         dbResult, ok = tmp.(shardIndexMapItem)
702                 }
703                 if !ok {
704                         err = kc.storage.Get(kc.tables.mapTable, shardKey,
                            ↪ &dbResult)
705                         if err == nil {
706                                 ctx.CacheSet(cacheKey, dbResult)
```

```
707                         }
708                 }
709         }
710         if err == nil {
711                 position := searchLevelOrder(len(dbResult.Keys), func(i
                    ↪  int) bool {
712                         return dbResult.Keys[i].Encoded >= key
713                 })
714                 var key shardIndexMapKey
715                 err = errNotFound
716                 if position < len(dbResult.Keys) {
717                         key = dbResult.Keys[position]
718                         if dbResult.Fields[key.Field] == mapKey.Field &&
                            ↪  key.Value == mapKey.Value {
719                                 err = nil
720                         }
721                 }
722                 if err == nil {
723                         _, err =
                            ↪  bitmap.FromBuffer(dbResult.Items[key.Start:key.End])
724                 }
725         }
726         return err
727 }
728 func (kc shardKVController) GetTreeNode(ctx ContextGetter, name string,
    ↪  nodeID uint32, result *TreeNode) error {
729         shards, err := getNumberShards(ctx, treeShards+"-"+name)
730         if err == nil {
731                 err = errNotFound
732                 l := len(shards.shards)
733                 if l > 0 && nodeID >= shards.shards[0] && nodeID <=
                    ↪  shards.shards[l-1] {
734                         err = nil
735                 }
736         }
737         var shardKey string
738         var shardID uint32
739         if err == nil {
```

```
740                 shardIndex := sort.Search(len(shards.shards)-1, func(i
                    ↪  int) bool {
741                         return shards.shards[i] > nodeID
742                 }) - 1
743                 shardID = shards.shards[shardIndex]
744                 shard := name + "-" + shards.encoded[shardIndex]
745                 shardKey = kc.getShardKey(ctx, shard)
746         }
747         var dbResult shardIndexTree
748         if err == nil {
749                 cacheKey := kc.tables.treeTable + "-" + shardKey
750                 tmp, ok := ctx.CacheGet(cacheKey)
751                 if ok {
752                         dbResult, ok = tmp.(shardIndexTree)
753                 }
754                 if !ok {
755                         err = kc.storage.Get(kc.tables.treeTable,
                            ↪  shardKey, &dbResult)
756                         if err == nil {
757                                 ctx.CacheSet(cacheKey, dbResult)
758                         }
759                 }
760         }
761         if err == nil {
762                 position := nodeID - shardID
763                 var key shardIndexTreeNode
764                 err = errNotFound
765                 if int(position) < len(dbResult.Nodes) {
766                         key = dbResult.Nodes[position]
767                         if uint32(key.ID) == nodeID {
768                                 err = nil
769                         }
770                 }
771                 if err == nil {
772                         result.Final = key.Final
773                         edgesLen := key.ChildrenEnd - key.ChildrenStart
774                         resultCap := cap(result.Edges)
775                         if resultCap < edgesLen {
```

```
776                                 result.Edges = append(result.Edges,
                                    ↪ make([]TreeEdge,
                                    ↪ edgesLen-len(result.Edges))...)
777                         }
778                         result.Edges = result.Edges[:edgesLen]
779                         for i := key.ChildrenStart; i < key.ChildrenEnd;
                            ↪ i++ {
780                                 edge := dbResult.Edges[i]
781                                 result.Edges[i-key.ChildrenStart] =
                                    ↪ TreeEdge{
782                                         Key:   edge.Key,
783                                         Value: uint32(edge.Value),
784                                 }
785                         }
786                 }
787         }
788         return err
789 }
790
791 func (kc shardKVController) GetDataMeta(ctx ContextGetter, itemKey string,
    ↪ result *DataMeta) error {
792         shards, err := getStringShards(ctx, dataMetaShards)
793         if err == nil {
794                 err = errNotFound
795                 l := len(shards)
796                 if l > 0 && itemKey >= shards[0] && itemKey <=
                    ↪ shards[l-1] {
797                         err = nil
798                 }
799         }
800         var shardKey string
801         if err == nil {
802                 shardIndex := (sort.Search((len(shards)-1)/2, func(i int)
                    ↪ bool {
803                         return shards[i*2] > itemKey
804                 }) * 2) - 1
805                 shardKey = kc.getShardKey(ctx, shards[shardIndex])
806         }
807         var dbResult shardIndexDataMeta
```

```
808         if err == nil {
809                 cacheKey := kc.tables.dataMetaTable + "-" + shardKey
810                 tmp, ok := ctx.CacheGet(cacheKey)
811                 if ok {
812                         dbResult, ok = tmp.(shardIndexDataMeta)
813                 }
814                 if !ok {
815                         err = kc.storage.Get(kc.tables.dataMetaTable,
                        ↪  shardKey, &dbResult)
816                         if err == nil {
817                                 ctx.CacheSet(cacheKey, dbResult)
818                         }
819                 }
820         }
821         if err == nil {
822                 position := searchLevelOrder(len(dbResult.Items), func(i
                ↪  int) bool {
823                         return dbResult.Items[i].Name >= itemKey
824                 })
825                 var item shardIndexDataMetaItem
826                 err = errNotFound
827                 if position < len(dbResult.Items) {
828                         item = dbResult.Items[position]
829                         if item.Name == itemKey {
830                                 err = nil
831                         }
832                 }
833                 if err == nil {
834                         err = kc.saveDataMetaResult(dbResult, item,
                        ↪  roaring.New(), result)
835                 }
836         }
837         return err
838 }
839
840 func (kc shardKVController) saveDataMetaResult(dbResult
    ↪  shardIndexDataMeta, item shardIndexDataMetaItem, bitmap
    ↪  *roaring.Bitmap, result *DataMeta) error {
841         result.Counter = uint32(item.Counter)
```

```
842         result.Deleted = item.Deleted
843         _, err :=
            ↪  bitmap.FromBuffer(dbResult.KeysIDs[item.Start:item.End])
844         if err == nil {
845                 result.Keys = make([]MapKey, bitmap.GetCardinality())
846                 it := bitmap.Iterator()
847                 i := 0
848                 for it.HasNext() {
849                         index := it.Next()
850                         result.Keys[i] = dbResult.Keys[index]
851                         i++
852                 }
853         }
854         return err
855 }
856
857 func (kc shardKVController) GetData(ctx ContextGetter, counter uint32,
    ↪  data interface{}) error {
858         shards, err := getNumberShards(ctx, dataShards)
859         if err == nil {
860                 err = errNotFound
861                 l := len(shards.shards)
862                 if l > 0 && counter >= shards.shards[0] && counter <=
                    ↪  shards.shards[l-1] {
863                         err = nil
864                 }
865         }
866         var shardKey string
867         var shard uint32
868         if err == nil {
869                 shardIndex := sort.Search(len(shards.shards)-1, func(i
                    ↪  int) bool {
870                         return shards.shards[i] > counter
871                 }) - 1
872                 shard = shards.shards[shardIndex]
873                 shardKey = kc.getShardKey(ctx,
                    ↪  shards.encoded[shardIndex])
874         }
875         var dbResult shardIndexData
```

```
876         if err == nil {
877                 cacheKey := kc.tables.dataTable + "-" + shardKey
878                 tmp, ok := ctx.CacheGet(cacheKey)
879                 if ok {
880                         dbResult, ok = tmp.(shardIndexData)
881                 }
882                 if !ok {
883                         err = kc.storage.Get(kc.tables.dataTable,
                        ↪  shardKey, &dbResult)
884                         if err == nil {
885                                 ctx.CacheSet(cacheKey, shardKey)
886                         }
887                 }
888         }
889         if err == nil {
890                 position := counter - shard
891                 err = errNotFound
892                 if int(position) < len(dbResult.Keys) {
893                         key := dbResult.Keys[position]
894                         if key == int32(counter) {
895                                 err = nil
896                                 reader := bytes.NewReader(dbResult.Data)
897                                 dec :=
                                ↪  kc.dependencies.GetStreamingDecoder(reader)
898                                 for i := uint32(0); err == nil && i <
                                ↪  position; i++ {
899                                     err = dec.Ignore()
900                                 }
901                                 if err == nil {
902                                         err = dec.Decode(data)
903                                 }
904                         }
905                 }
906         }
907         return err
908 }
909
910 func (kc shardKVController) deleteTable(ctx ContextGetter, table string,
    ↪  result interface{}, toPreserve bool, values []string) error {
```

```
911         var err error
912         it := kc.storage.GetAllKeys(table)
913         for err == nil {
914                 var key string
915                 key, err = it.Next(&result)
916                 if err == nil && strings.HasPrefix(key, ctx.GetVersion())
                    ↪ {
917                         hasPrefix := false
918                         for _, check := range values {
919                                 if strings.HasPrefix(key, check) {
920                                         hasPrefix = true
921                                         break
922                                 }
923                         }
924                         if !hasPrefix == toPreserve {
925                                 err = kc.storage.Del(table, key)
926                         }
927                 }
928         }
929         if kv.IsDoneError(err) {
930                 err = nil
931         }
932         return err
933 }
934
935 func (kc shardKVController) Delete(ctx ContextGetter, toPreserve bool,
    ↪  values []string) error {
936         // Used by Delete and RunGC methods in Index
937         err := kc.deleteTable(ctx, kc.tables.mapTable,
            ↪  &shardIndexMapItem{}, toPreserve, values)
938         if err == nil {
939                 err = kc.deleteTable(ctx, kc.tables.treeTable,
                    ↪  &shardIndexTree{}, toPreserve, values)
940         }
941         if err == nil {
942                 err = kc.deleteTable(ctx, kc.tables.dataMetaTable,
                    ↪  &shardIndexDataMeta{}, toPreserve, values)
943         }
944         if err == nil {
```

```
945                 err = kc.deleteTable(ctx, kc.tables.dataTable,
                    ↪  &shardIndexData{}, toPreserve, values)
946         }
947         return err
948 }
949
950 type ShardDependencies interface {
951         GetStreamingDecoder(io.Reader) coder.StreamingDecoder
952         GetStreamingEncoder(io.WriteCloser) coder.StreamingEncoder
953 }
954
955 func NewShardKVController(storage kv.Store, dependencies
    ↪  ShardDependencies, options ShardKVOptions) Controller {
956         options = getDefaultShardKVOptions(options)
957         return shardKVController{
958                 storage:      storage,
959                 options:      options,
960                 dependencies: dependencies,
961                 tables: shardKVTables{
962                         dataMetaTable: options.TablePrefix +
                            ↪  "-data-meta",
963                         dataTable:     options.TablePrefix + "-data",
964                         mapTable:      options.TablePrefix + "-map",
965                         treeTable:     options.TablePrefix + "-tree",
966                 },
967         }
968 }
```

Arquivo shard_controller_inserter.go

```
1  package indexer
2
3  import (
4          "bytes"
5          "errors"
6          "math"
7          "sort"
8          "strconv"
9          "sync"
10
```

```
11          "github.com/RoaringBitmap/roaring"
12          "github.com/fjorgemota/gurudamatricula/util/coder"
13  )
14
15  type shardKVInserter struct {
16          ctx         ContextSetter
17          controller  shardKVController
18          lock        sync.Mutex
19          dataKeys    []int32
20          shardsBuf   *bytes.Buffer
21          dataBuf     *shardBuf
22          dataEncoder coder.StreamingEncoder
23          lastCounter uint32
24          err         error
25  }
26
27  func (ki *shardKVInserter) addDataShard() {
28          var id string
29          shard := ki.dataKeys[0]
30          id, ki.err = ki.controller.options.IntEncoder.Encode(int(shard))
31          if ki.err == nil {
32                  ki.err = ki.dataEncoder.Flush()
33          }
34          if ki.err == nil {
35                  ki.err = ki.dataEncoder.Close()
36          }
37          if ki.err == nil {
38                  key := ki.controller.getShardKey(ki.ctx, id)
39                  ki.err =
                    ↪  ki.controller.storage.Set(ki.controller.tables.dataTable,
                    ↪  key, &shardIndexData{
40                          Data: ki.dataBuf.Bytes(),
41                          Keys: ki.dataKeys,
42                  })
43                  ki.dataBuf.Reset()
44                  ki.dataEncoder =
                    ↪  ki.controller.dependencies.GetStreamingEncoder(ki.dataBuf)
45                  ki.dataKeys = ki.dataKeys[:0]
46          }
```

```
47          if ki.err == nil {
48                  if ki.shardsBuf.Len() > 0 {
49                          ki.err = ki.shardsBuf.WriteByte('|')
50                  }
51                  if ki.err == nil {
52                          _, ki.err =
                            ↪  ki.shardsBuf.WriteString(strconv.Itoa(int(shard)))
53                  }
54                  if ki.err == nil {
55                          ki.err = ki.shardsBuf.WriteByte('|')
56                  }
57                  if ki.err == nil {
58                          _, ki.err = ki.shardsBuf.WriteString(id)
59                  }
60          }
61  }
62
63  func (ki *shardKVInserter) AddData(counter uint32, data interface{})
    ↪  error {
64          ki.lock.Lock()
65          defer ki.lock.Unlock()
66          oldLen := ki.dataBuf.Len()
67          if ki.err == nil {
68                  ki.err = ki.dataEncoder.Encode(data)
69          }
70          if ki.err == nil {
71                  ki.err = ki.dataEncoder.Flush()
72          }
73          if ki.err == nil && ki.dataBuf.Len() >=
            ↪  ki.controller.options.MaximumSize && oldLen > 0 {
74                  ki.dataBuf.Truncate(oldLen)
75                  ki.addDataShard()
76                  if ki.err == nil {
77                          ki.err = ki.dataEncoder.Encode(data)
78                  }
79                  if ki.err == nil {
80                          ki.err = ki.dataEncoder.Flush()
81                  }
82          }
```

```
83          if ki.err == nil && ki.dataBuf.Len() >=
            ↪  ki.controller.options.MaximumSize {
84                  ki.err = errors.New("indexer: Entity is bigger than
                    ↪  maximum size defined")
85          }
86          if ki.err == nil {
87                  ki.dataKeys = append(ki.dataKeys, int32(counter))
88                  ki.lastCounter = counter
89          }
90          return ki.err
91  }
92
93  func (ki *shardKVInserter) addMapShard(shardID int, mapResult
    ↪  shardIndexMapItem) error {
94          if ki.err == nil {
95                  ki.err = addMapShard(ki.controller, shardOptions{
96                          dataBuf:   ki.dataBuf,
97                          shardsBuf: ki.shardsBuf,
98                          shardID:   shardID,
99                          ctx:       ki.ctx,
100                 }, mapResult)
101         }
102         return ki.err
103 }
104
105 func (ki *shardKVInserter) CommitMap(bitmaps map[string]Bitmap, keys
    ↪  map[MapKey]Map) error {
106         ki.lock.Lock()
107         defer ki.lock.Unlock()
108         keyList := make(shardedEncodedMapKeySorter, len(keys))
109         for mapKey, value := range keys {
110                 key := ki.controller.options.Encoder(mapKey)
111                 keyList[value.Index].Encoded = key
112                 keyList[value.Index].Key = mapKey
113         }
114         sort.Sort(keyList)
115         mapItemRange := make(map[string]shardRange)
116         mapFields := make(map[string]int)
117         var totalSize int
```

```
118             totalSize += 32 // Overhead per record
119             totalSize += 7  // Fields (name)
120             totalSize += 1  // Fields overhead
121             totalSize += 5  // Keys (name)
122             totalSize += 1  // Keys overhead
123             totalSize += 6  // Items (name)
124             totalSize += 1  // Items overhead
125             var lastKey string
126             var mapResult shardIndexMapItem
127             var shardID int
128             for _, itemKey := range keyList {
129                     if totalSize+ki.dataBuf.Len() >=
                        ↪  ki.controller.options.MaximumSize && ki.err == nil {
130                             lastKey =
                                ↪  mapResult.Keys[len(mapResult.Keys)-1].Encoded
131                             ki.err = ki.addMapShard(shardID, mapResult)
132                             mapResult.Items = mapResult.Items[:0]
133                             mapResult.Fields = mapResult.Fields[:0]
134                             mapResult.Keys = mapResult.Keys[:0]
135                             totalSize = 32 // Overhead per record
136                             totalSize += 7 // Fields (name)
137                             totalSize += 1 // Fields overhead
138                             totalSize += 5 // Keys (name)
139                             totalSize += 1 // Keys overhead
140                             totalSize += 6 // Items (name)
141                             totalSize += 1 // Items overhead
142                             for key := range mapItemRange {
143                                     delete(mapItemRange, key)
144                             }
145                             for key := range mapFields {
146                                     delete(mapFields, key)
147                             }
148                             ki.dataBuf.Reset()
149                             shardID++
150                     }
151                     var ok bool
152                     var itemRange shardRange
153                     mapKey := itemKey.Key
154                     item := keys[mapKey]
```

```
155                 if itemRange, ok = mapItemRange[item.Bitmap]; !ok {
156                         bitmap := bitmaps[item.Bitmap].Bitmap
157                         if bitmap.IsEmpty() {
158                                 // Ignore empty bitmaps because there's
                                    ↪  no datameta
159                                 // reffering to this information
160                                 continue
161                         }
162                         start := ki.dataBuf.Len()
163                         if ki.err == nil {
164                                 _, ki.err = bitmap.WriteTo(ki.dataBuf)
165                         }
166                         itemRange = shardRange{
167                                 Start: start,
168                                 End:   ki.dataBuf.Len(),
169                         }
170                         mapItemRange[item.Bitmap] = itemRange
171                 }
172                 var fieldIndex int
173                 if fieldIndex, ok = mapFields[mapKey.Field]; !ok {
174                         totalSize += len(mapKey.Field) + 1
175                         mapResult.Fields = append(mapResult.Fields,
                            ↪  mapKey.Field)
176                         fieldIndex = len(mapResult.Fields) - 1
177                         mapFields[mapKey.Field] = fieldIndex
178                 }
179                 totalSize += 8                          // Encoded (name)
180                 totalSize += len(itemKey.Encoded) + 1 // Encoded (value)
181                 totalSize += 6                          // Field (name)
182                 totalSize += 8                          // Field (value)
183                 totalSize += 6                          // Value (name)
184                 totalSize += 8                          // Value (value)
185                 totalSize += 6                          // Start (name)
186                 totalSize += 8                          // Start (value)
187                 totalSize += 4                          // End (name)
188                 totalSize += 8                          // End (value)
189                 mapResult.Keys = append(mapResult.Keys, shardIndexMapKey{
190                         Encoded: itemKey.Encoded,
191                         Field:   fieldIndex,
```

```
192                         Value:   mapKey.Value,
193                         Start:   itemRange.Start,
194                         End:     itemRange.End,
195                 })
196                 if ki.err == nil {
197                         ki.err = errShardOverflow
198                         if shardID < math.MaxUint16 &&
                            ↪  (len(mapResult.Keys)-1) < math.MaxUint16 {
199                             ki.err = nil
200                         }
201                 }
202         }
203         if ki.err == nil && len(mapResult.Keys) > 0 {
204                 lastKey = mapResult.Keys[len(mapResult.Keys)-1].Encoded
205                 ki.err = ki.addMapShard(shardID, mapResult)
206         }
207         if ki.err == nil && lastKey != "" {
208                 ki.err = ki.shardsBuf.WriteByte('|')
209                 if ki.err == nil {
210                         _, ki.err = ki.shardsBuf.WriteString(lastKey)
211                 }
212         }
213         ki.ctx.Set(mapShards, ki.shardsBuf.String())
214         ki.shardsBuf.Reset()
215         ki.dataBuf.Reset()
216         return ki.err
217 }
218
219 func (ki *shardKVInserter) CommitData() error {
220         ki.lock.Lock()
221         defer ki.lock.Unlock()
222         if len(ki.dataKeys) > 0 {
223                 // There things to process yet..add new shard
224                 ki.addDataShard()
225         }
226         if ki.shardsBuf.Len() > 0 {
227                 ki.err = ki.shardsBuf.WriteByte('|')
228                 if ki.err == nil {
```

```
229                         _, ki.err =
                            ↪  ki.shardsBuf.WriteString(strconv.Itoa(int(ki.lastCounter)))
230                 }
231         }
232         ki.ctx.Set(dataShards, ki.shardsBuf.String())
233         // We just reset dataBuf to it's be empty because
234         // we will reuse the same structure throught the code
235         ki.shardsBuf.Reset()
236         ki.dataBuf.Reset()
237         return ki.err
238 }
239
240 func (ki *shardKVInserter) addTreeShard(name string, shardID int, tree
    ↪  shardIndexTree) error {
241         firstKey := tree.Nodes[0].ID
242         id, err := ki.controller.options.IntEncoder.Encode(shardID)
243         if err == nil {
244                 shardKey := name + "-" + id
245                 shardKey = ki.controller.getShardKey(ki.ctx, shardKey)
246                 err =
                    ↪  ki.controller.storage.Set(ki.controller.tables.treeTable,
                    ↪  shardKey, &tree)
247         }
248         if err == nil {
249                 if ki.shardsBuf.Len() > 0 {
250                         err = ki.shardsBuf.WriteByte('|')
251                 }
252                 if err == nil {
253                         _, err =
                            ↪  ki.shardsBuf.WriteString(strconv.Itoa(int(firstKey)))
254                 }
255                 if err == nil {
256                         err = ki.shardsBuf.WriteByte('|')
257                 }
258                 if err == nil {
259                         _, err = ki.shardsBuf.WriteString(id)
260                 }
261         }
262         return err
```

```
263 }
264
265 func (ki *shardKVInserter) CommitTree(name string, tree []TreeNode) error
    ↪ {
266         var treeResult shardIndexTree
267         var totalSize int
268         totalSize += 32 // Overhead per record
269         totalSize += 6  // Nodes (name)
270         totalSize += 1  // Nodes overhead
271         totalSize += 6  // Edges (name)
272         totalSize += 1  // Edges overhead
273         var shardID int
274         var lastKey int32
275         for _, node := range tree {
276                 if totalSize >= ki.controller.options.MaximumSize &&
                    ↪  ki.err == nil {
277                         ki.err = ki.addTreeShard(name, shardID,
                            ↪  treeResult)
278                         lastKey =
                            ↪  treeResult.Nodes[len(treeResult.Nodes)-1].ID
279                         treeResult.Nodes = treeResult.Nodes[:0]
280                         treeResult.Edges = treeResult.Edges[:0]
281                         totalSize = 32 // Overhead per record
282                         totalSize += 6 // Nodes (name)
283                         totalSize += 1 // Nodes overhead
284                         totalSize += 6 // Edges (name)
285                         totalSize += 1 // Edges overhead
286                         ki.dataBuf.Reset()
287                         shardID++
288                 }
289                 if ki.err == nil {
290                         totalSize += 3  // ID (name)
291                         totalSize += 8  // ID (value)
292                         totalSize += 6  // Final (name)
293                         totalSize += 1  // Final (value)
294                         totalSize += 14 // ChildrenStart (name)
295                         totalSize += 8  // ChildrenStart (value)
296                         totalSize += 12 // ChildrenEnd (name)
297                         totalSize += 8  // ChildrenEnd (value)
```

```
298                         treeResult.Nodes = append(treeResult.Nodes,
                            ↪  shardIndexTreeNode{
299                                 ID:            int32(node.ID),
300                                 Final:         node.Final,
301                                 ChildrenStart: len(treeResult.Edges),
302                                 ChildrenEnd:   len(treeResult.Edges) +
                                    ↪  len(node.Edges),
303                         })
304                 }
305                 for _, edge := range node.Edges {
306                         totalSize += 4 // Key (name)
307                         totalSize += 8 // Key (value)
308                         totalSize += 6 // Value (name)
309                         totalSize += 8 // Value (value)
310                         treeResult.Edges = append(treeResult.Edges,
                            ↪  shardIndexTreeEdge{
311                                 Key:   edge.Key,
312                                 Value: int32(edge.Value),
313                         })
314                 }
315         }
316         if ki.err == nil && len(treeResult.Nodes) > 0 {
317                 ki.err = ki.addTreeShard(name, shardID, treeResult)
318                 lastKey = treeResult.Nodes[len(treeResult.Nodes)-1].ID
319         }
320         if ki.err == nil && ki.shardsBuf.Len() > 0 {
321                 ki.err = ki.shardsBuf.WriteByte('|')
322                 if ki.err == nil {
323                         _, ki.err =
                            ↪  ki.shardsBuf.WriteString(strconv.Itoa(int(lastKey)))
324                 }
325         }
326         ki.ctx.Set(treeShards+"-"+name, ki.shardsBuf.String())
327         // we will reuse the same structure throught the code
328         ki.shardsBuf.Reset()
329         return ki.err
330 }
331
```

```
332 func (ki *shardKVInserter) addDataMetaShard(shardID int, dataMetaResult
    ↪  shardIndexDataMeta) error {
333         if ki.err == nil {
334                 ki.err = addDataMetaShard(ki.controller, shardOptions{
335                         dataBuf:   ki.dataBuf,
336                         shardsBuf: ki.shardsBuf,
337                         shardID:   shardID,
338                         ctx:       ki.ctx,
339                 }, dataMetaResult)
340         }
341         return ki.err
342 }
343
344 func (ki *shardKVInserter) CommitDataMeta(keys map[MapKey]Map,
    ↪  dataMetaKeys map[string]InputDataMeta) error {
345         keyList := make([]string, 0, len(dataMetaKeys))
346         for key := range dataMetaKeys {
347                 keyList = append(keyList, key)
348         }
349         sort.Strings(keyList)
350         mapKeyList := make([]MapKey, len(keys))
351         for key, value := range keys {
352                 mapKeyList[value.Index] = key
353         }
354         var dataMetaResult shardIndexDataMeta
355         var totalSize int
356         var lastKey string
357         totalSize += 32 // Overhead per record
358         totalSize += 6  // Items (name)
359         totalSize += 1  // Items overhead
360         totalSize += 7  // KeyIDs (name)
361         totalSize += 1  // KeyIDs overhead
362         totalSize += 5  // Keys (name)
363         totalSize += 1  // Keys (overhead)
364         var shardID int
365         dataKeys := roaring.New()
366         mapKeyIndexes := make(map[MapKey]uint32)
367         for _, itemKey := range keyList {
```

```
368                 if totalSize+ki.dataBuf.Len() >=
                    ↪  ki.controller.options.MaximumSize && ki.err == nil {
369                         lastKey =
                            ↪  dataMetaResult.Items[len(dataMetaResult.Items)-1].Name
370                         ki.err = ki.addDataMetaShard(shardID,
                            ↪  dataMetaResult)
371                         dataMetaResult.Items = dataMetaResult.Items[:0]
372                         dataMetaResult.KeysIDs =
                            ↪  dataMetaResult.KeysIDs[:0]
373                         dataMetaResult.Keys = dataMetaResult.Keys[:0]
374                         for key := range mapKeyIndexes {
375                                 delete(mapKeyIndexes, key)
376                         }
377                         totalSize = 32 // Overhead per record
378                         totalSize += 6 // Items (name)
379                         totalSize += 1 // Items overhead
380                         totalSize += 7 // KeyIDs (name)
381                         totalSize += 1 // KeyIDs overhead
382                         totalSize += 5 // Keys (name)
383                         totalSize += 1 // Keys (overhead)
384                         ki.dataBuf.Reset()
385                         shardID++
386                 }
387                 item := dataMetaKeys[itemKey]
388                 it := item.Keys.Iterator()
389                 for it.HasNext() {
390                         index := it.Next()
391                         mapKey := mapKeyList[index]
392                         if _, ok := mapKeyIndexes[mapKey]; !ok {
393                                 dataMetaResult.Keys =
                                    ↪  append(dataMetaResult.Keys, mapKey)
394                                 mapKeyIndexes[mapKey] =
                                    ↪  uint32(len(dataMetaResult.Keys) - 1)
395                                 totalSize += 6                    //
                                    ↪  Field (name)
396                                 totalSize += len(mapKey.Field) + 1 //
                                    ↪  Field (value)
397                                 totalSize += 6                    //
                                    ↪  Value (name)
```

```
398                                 totalSize += len(mapKey.Value) + 1 //
                                    ↪  Value (value)
399                         }
400                         dataKeys.Add(mapKeyIndexes[mapKey])
401                 }
402                 if ki.err == nil {
403                         totalSize += 5                  // Name (name)
404                         totalSize += len(itemKey) + 1 // Name (value)
405                         totalSize += 8                  // Counter (name)
406                         totalSize += 8                  // Counter (value)
407                         totalSize += 8                  // Deleted (name)
408                         totalSize += 1                  // Deleted (value)
409                         totalSize += 6                  // Start (name)
410                         totalSize += 8                  // Start (value)
411                         totalSize += 4                  // End (name)
412                         totalSize += 8                  // End (value)
413                         dataKeys.RunOptimize()
414                         start := ki.dataBuf.Len()
415                         _, ki.err = dataKeys.WriteTo(ki.dataBuf)
416                         if ki.err == nil {
417                                 dataMetaResult.Items =
                                    ↪  append(dataMetaResult.Items,
                                    ↪  shardIndexDataMetaItem{
418                                         Name:    itemKey,
419                                         Counter: int32(item.Counter),
420                                         Deleted: item.Deleted,
421                                         Start:   start,
422                                         End:     ki.dataBuf.Len(),
423                                 })
424                         }
425                 }
426                 dataKeys.Clear()
427                 delete(dataMetaKeys, itemKey)
428         }
429         if ki.err == nil && len(dataMetaResult.Items) > 0 {
430                 lastKey =
                    ↪  dataMetaResult.Items[len(dataMetaResult.Items)-1].Name
431                 ki.err = ki.addDataMetaShard(shardID, dataMetaResult)
432         }
```

```
433         if ki.err == nil && ki.shardsBuf.Len() > 0 {
434                 ki.err = ki.shardsBuf.WriteByte('|')
435                 if ki.err == nil {
436                         _, ki.err = ki.shardsBuf.WriteString(lastKey)
437                 }
438         }
439         ki.ctx.Set(dataMetaShards, ki.shardsBuf.String())
440         ki.shardsBuf.Reset()
441         ki.dataBuf.Reset()
442         return ki.err
443 }
```

Arquivo shard_controller_utilities.go

```
1  package indexer
2
3  import (
4          "bytes"
5          "strings"
6          "sync"
7
8          "github.com/fjorgemota/gurudamatricula/util/pool"
9          "github.com/pkg/errors"
10 )
11
12 type shardOptions struct {
13         shardID int
14         dataBuf interface {
15                 Bytes() []byte
16         }
17         shardsBuf *bytes.Buffer
18         ctx       ContextSetter
19 }
20
21 func addMapShard(kc shardKVController, options shardOptions, mapResult
   ↪  shardIndexMapItem) error {
22         var firstKey string
23         firstKey = mapResult.Keys[0].Encoded
24         id, err := kc.options.IntEncoder.Encode(options.shardID)
25         if err == nil {
```

```
                shardKey := kc.getShardKey(options.ctx, id)
                mapResult.Items = options.dataBuf.Bytes()
                mapResult.Keys =
                ↪  toLevelOrderShardIndexMapKey(mapResult.Keys)
                err = kc.storage.Set(kc.tables.mapTable, shardKey,
                ↪  &mapResult)
        }
        if err == nil {
                if options.shardsBuf.Len() > 0 {
                        err = options.shardsBuf.WriteByte('|')
                }
                if err == nil {
                        _, err = options.shardsBuf.WriteString(firstKey)
                }
                if err == nil {
                        err = options.shardsBuf.WriteByte('|')
                }
                if err == nil {
                        _, err = options.shardsBuf.WriteString(id)
                }
        }
        return err
}

func addDataMetaShard(kc shardKVController, options shardOptions,
↪  dataMetaResult shardIndexDataMeta) error {
        firstKey := dataMetaResult.Items[0].Name
        id, err := kc.options.IntEncoder.Encode(options.shardID)
        if err == nil {
                shardKey := kc.getShardKey(options.ctx, id)
                dataMetaResult.Items =
                ↪  toLevelOrderShardIndexDataMetaItem(dataMetaResult.Items)
                dataMetaResult.KeysIDs = options.dataBuf.Bytes()
                err = kc.storage.Set(kc.tables.dataMetaTable, shardKey,
                ↪  &dataMetaResult)
        }
        if err == nil {
                if options.shardsBuf.Len() > 0 {
                        err = options.shardsBuf.WriteByte('|')
```

```
                }
                if err == nil {
                        _, err = options.shardsBuf.WriteString(firstKey)
                }
                if err == nil {
                        err = options.shardsBuf.WriteByte('|')
                }
                if err == nil {
                        _, err = options.shardsBuf.WriteString(id)
                }
        }
        return err
}

func getStringShards(ctx ContextGetter, shardsConfig string) ([]string,
↪  error) {
        var result []string
        var err error
        tmp, ok := ctx.CacheGet(shardsConfig)
        if ok {
                result, ok = tmp.([]string)
        }
        if !ok {
                shardList := ctx.Get(shardsConfig)
                if len(shardList) > 0 {
                        result = strings.Split(shardList, "|")
                }
                if err == nil {
                        ctx.CacheSet(shardsConfig, result)
                }
        }
        return result, err
}

type numberShards struct {
        shards  []uint32
        encoded []string
}

```

```
98  func getNumberShards(ctx ContextGetter, shardConfig string) (numberShards,
    ↪  error) {
99          var result numberShards
100         var err error
101         tmp, ok := ctx.CacheGet(shardConfig)
102         if ok {
103                 result, ok = tmp.(numberShards)
104         }
105         if !ok {
106                 shardBytes := []byte(ctx.Get(shardConfig))
107                 if len(shardBytes) > 0 {
108                         converter := pool.GetAtoiCache()
109                         parts := bytes.Split(shardBytes, []byte("|"))
110                         for i, part := range parts {
111                                 if i%2 == 0 {
112                                         var shard int
113                                         if err == nil {
114                                                 shard, err =
                                                ↪  converter.Atoi(part)
115                                         }
116                                         if err == nil {
117                                                 result.shards =
                                                ↪  append(result.shards,
                                                ↪  uint32(shard))
118                                         }
119                                 } else {
120                                         result.encoded =
                                                ↪  append(result.encoded,
                                                ↪  string(part))
121                                 }
122                         }
123                         pool.PutAtoiCache(converter)
124                         err = errors.Wrapf(err, "error while processing
                        ↪  config '%s' with value '%s'", shardConfig,
                        ↪  shardBytes)
125                 }
126                 if err == nil {
127                         ctx.CacheSet(shardConfig, result)
128                 }
```

```
129         }
130         return result, err
131 }
132
133 func root(n int) int {
134         if n <= 1 {
135                 return 0
136         }
137         i := 2
138         for i <= n {
139                 i *= 2
140         }
141         result1 := i/2 - 1
142         result2 := n - i/4
143         if result1 < result2 {
144                 return result1
145         }
146         return result2
147 }
148
149 type queueItemLevelOrder struct {
150         index    int
151         length   int
152         position int
153 }
154
155 var queueItemPool = sync.Pool{
156         New: func() interface{} {
157                 return make([]queueItemLevelOrder, 0, 32)
158         },
159 }
160
161 func levelOrder(length int, swap func(originalIndex, levelOrderIndex
    ↪  int)) {
162         queue := queueItemPool.Get().([]queueItemLevelOrder)
163         queue = append(queue, queueItemLevelOrder{
164                 index:    0,
165                 length:   length,
166                 position: 0,
```

```
167         })
168         for len(queue) > 0 {
169                 var item queueItemLevelOrder
170                 item, queue = queue[len(queue)-1], queue[:len(queue)-1]
171                 if item.length > 0 {
172                         h := root(item.length)
173                         swap(item.position+h, item.index)
174                         if h > 0 {
175                                 queue = append(queue,
                                    ↪ queueItemLevelOrder{
176                                         index:    2*item.index + 1,
177                                         length:   h,
178                                         position: item.position,
179                                 })
180                         }
181                         if item.length-h-1 > 0 {
182                                 queue = append(queue,
                                    ↪ queueItemLevelOrder{
183                                         index:    2*item.index + 2,
184                                         length:   item.length - h - 1,
185                                         position: item.position + h + 1,
186                                 })
187                         }
188                 }
189         }
190         queueItemPool.Put(queue[:0])
191 }
192
193 func searchLevelOrder(n int, fn func(i int) bool) int {
194         i := n
195         j := 0
196         for j < n {
197                 if !fn(j) {
198                         j = 2*j + 2
199                 } else {
200                         i = j
201                         j = 2*j + 1
202                 }
203         }
```

```
204         return i
205 }
206
207 func toLevelOrderShardIndexMapKey(original []shardIndexMapKey)
    ↪  []shardIndexMapKey {
208         result := make([]shardIndexMapKey, len(original))
209         levelOrder(len(original), func(originalIndex, levelOrderIndex
            ↪  int) {
210                 result[levelOrderIndex] = original[originalIndex]
211         })
212         return result
213 }
214
215 func fromLevelOrderShardIndexMapKey(levelOrdered []shardIndexMapKey)
    ↪  []shardIndexMapKey {
216         result := make([]shardIndexMapKey, len(levelOrdered))
217         levelOrder(len(levelOrdered), func(originalIndex, levelOrderIndex
            ↪  int) {
218                 result[originalIndex] = levelOrdered[levelOrderIndex]
219         })
220         return result
221 }
222
223 func toLevelOrderShardIndexDataMetaItem(original
    ↪  []shardIndexDataMetaItem) []shardIndexDataMetaItem {
224         result := make([]shardIndexDataMetaItem, len(original))
225         levelOrder(len(original), func(originalIndex, levelOrderIndex
            ↪  int) {
226                 result[levelOrderIndex] = original[originalIndex]
227         })
228         return result
229 }
230
231 func fromLevelOrderShardIndexDataMetaItem(levelOrdered
    ↪  []shardIndexDataMetaItem) []shardIndexDataMetaItem {
232         result := make([]shardIndexDataMetaItem, len(levelOrdered))
233         levelOrder(len(levelOrdered), func(originalIndex, levelOrderIndex
            ↪  int) {
234                 result[originalIndex] = levelOrdered[levelOrderIndex]
```

```
        })
        return result
}

func cleanShardIndexMapItem(item shardIndexMapItem) shardIndexMapItem {
        for i := range item.Fields {
                item.Fields[i] = ""
        }
        item.Fields = item.Fields[:0]
        for i := range item.Keys {
                item.Keys[i] = shardIndexMapKey{}
        }
        item.Keys = item.Keys[:0]
        item.Items = item.Items[:0]
        return item
}

func cleanShardIndexDataMeta(item shardIndexDataMeta) shardIndexDataMeta
↪ {
        for i := range item.Items {
                item.Items[i] = shardIndexDataMetaItem{}
        }
        item.Items = item.Items[:0]
        for i := range item.Keys {
                item.Keys[i] = MapKey{}
        }
        item.Keys = item.Keys[:0]
        item.KeysIDs = item.KeysIDs[:0]
        return item
}

func clearShardIndexData(item shardIndexData) shardIndexData {
        item.Data = item.Data[:0]
        item.Keys = item.Keys[:0]
        return item
}
```

Arquivo shard_sorter.go

```
package indexer
```

```
2
3  type shardedEncodedMapKey struct {
4          Encoded string
5          Key     MapKey
6  }
7  type shardedEncodedMapKeySorter []shardedEncodedMapKey
8
9  func (semks shardedEncodedMapKeySorter) Less(i, j int) bool {
10         return semks[i].Encoded < semks[j].Encoded
11 }
12
13 func (semks shardedEncodedMapKeySorter) Len() int {
14         return len(semks)
15 }
16
17 func (semks shardedEncodedMapKeySorter) Swap(i, j int) {
18         semks[i], semks[j] = semks[j], semks[i]
19 }
```

Arquivo utilities.go

```
1  package indexer
2
3  import (
4          "bytes"
5          "sort"
6          "sync"
7
8          "github.com/RoaringBitmap/roaring"
9          "github.com/fjorgemota/gurudamatricula/util/indexer/stringset"
10         "github.com/willf/bloom"
11 )
12
13 var mapPool = sync.Pool{
14         New: func() interface{} {
15                 return make(map[string]struct{}, 0)
16         },
17 }
18
19 var zero struct{}
```

```
20
21 func getBloomFilter(bs []byte) (*bloom.BloomFilter, error) {
22         filter := &bloom.BloomFilter{}
23         var err error
24         if bs != nil {
25                 reader := bytes.NewReader(bs)
26                 _, err = filter.ReadFrom(reader)
27         }
28         return filter, err
29 }
30
31 func getDataFilters(old exactMetaVersions) ([]*bloom.BloomFilter, error)
   ↪ {
32         result := make([]*bloom.BloomFilter, len(old.Versions))
33         var err error
34         for i, version := range old.Versions {
35                 if err == nil {
36                         result[i], err =
                           ↪ getBloomFilter(version.AllDataKeys)
37                 }
38         }
39         return result, err
40 }
41
42 func getMapFilters(old exactMetaVersions) ([]*bloom.BloomFilter, error) {
43         result := make([]*bloom.BloomFilter, len(old.Versions))
44         var err error
45         for i, version := range old.Versions {
46                 if err == nil {
47                         result[i], err =
                           ↪ getBloomFilter(version.AllMapKeys)
48                 }
49         }
50         return result, err
51 }
52
53 func getItemsBitmaps(old exactMetaVersions) ([]*roaring.Bitmap, error) {
54         result := make([]*roaring.Bitmap, len(old.Versions))
55         var err error
```

```
56          for i, version := range old.Versions {
57                  if err == nil {
58                          result[i] = roaring.NewBitmap()
59                  }
60                  if err == nil && version.Items != nil {
61                          _, err = result[i].FromBuffer(version.Items)
62                  }
63          }
64          return result, err
65  }
66
67  func cleanUnique(uniqueify map[string]struct{}) map[string]struct{} {
68          for key := range uniqueify {
69                  delete(uniqueify, key)
70          }
71          return uniqueify
72  }
73
74  type ftTreeEdgeSorter []TreeEdge
75
76  func (ftes ftTreeEdgeSorter) Len() int {
77          return len(ftes)
78  }
79
80  func (ftes ftTreeEdgeSorter) Swap(i, j int) {
81          ftes[i], ftes[j] = ftes[j], ftes[i]
82  }
83
84  func (ftes ftTreeEdgeSorter) Less(i, j int) bool {
85          return ftes[i].Key < ftes[j].Key
86  }
87
88  func getNodes(tree *stringset.Builder) []TreeNode {
89          mapping := tree.GetNodes()
90          nodes := make([]TreeNode, len(mapping))
91          for elem, index := range mapping {
92                  edges := elem.Edges()
93                  nodes[index] = TreeNode{
94                          ID:    uint32(index),
```

```
95                          Final: elem.Final(),
96                          Edges: make([]TreeEdge, len(edges)),
97                  }
98                  count := 0
99                  for key, edge := range edges {
100                         nodes[index].Edges[count] = TreeEdge{
101                                 Key:   key,
102                                 Value: uint32(mapping[edge]),
103                         }
104                         count++
105                 }
106                 sort.Sort(ftTreeEdgeSorter(nodes[index].Edges))
107         }
108         return nodes
109 }
```

B.1.27 Pasta util/kv

Arquivo appengine.go

```
1  package kv
2
3  import (
4          "fmt"
5          "reflect"
6          "sync"
7
8          "golang.org/x/net/context"
9          "google.golang.org/appengine"
10         "google.golang.org/appengine/datastore"
11 )
12
13 // appEngineIterator is an iterator for AppEngine queries
14 type appEngineIterator struct {
15         it   *datastore.Iterator
16         lock sync.Mutex
17 }
18
19 // Next will return the next result available for the AppEngine query
20 func (i *appEngineIterator) Next(result interface{}) (string, error) {
```

```
21          i.lock.Lock()
22          defer i.lock.Unlock()
23          var err error
24          if reflect.ValueOf(result).Kind() != reflect.Ptr {
25                  err = errResultMustBeAPointer
26          }
27          var k *datastore.Key
28          if err == nil {
29                  k, err = i.it.Next(result)
30          }
31          if err != nil {
32                  if err == datastore.Done {
33                          // Remap error from datastore.Done to ErrDone
34                          err = errDone
35                  }
36                  return "", err
37          }
38          return k.StringID(), err
39  }
40
41  type appEngineQuery struct {
42          q   *datastore.Query
43          ctx context.Context
44  }
45
46  func (q appEngineQuery) Filter(field, comparator string, value
    ↪  interface{}) Query {
47          return appEngineQuery{
48                  q:   q.q.Filter(field+" "+comparator, value),
49                  ctx: q.ctx,
50          }
51  }
52
53  func (q appEngineQuery) KeysOnly() Query {
54          return appEngineQuery{
55                  q:   q.q.KeysOnly(),
56                  ctx: q.ctx,
57          }
58  }
```

```
59
60 func (q appEngineQuery) Run() Iterator {
61         return &appEngineIterator{
62                 it: q.q.Run(q.ctx),
63         }
64 }
65
66 type appEngineBatchItem struct {
67         keys   []*datastore.Key
68         values reflect.Value
69 }
70 type appEngineBatch struct {
71         items map[reflect.Type]*appEngineBatchItem
72         c     context.Context
73         lock  sync.Mutex
74 }
75
76 func (b *appEngineBatch) Set(table, key string, value interface{}) error
   ↪ {
77         b.lock.Lock()
78         defer b.lock.Unlock()
79         k := datastore.NewKey(b.c, table, key, 0, nil)
80         v := reflect.ValueOf(value)
81         t := v.Type()
82         item, ok := b.items[t]
83         if !ok {
84                 item = &appEngineBatchItem{
85                         values: reflect.MakeSlice(reflect.SliceOf(t), 0,
                           ↪  1),
86                 }
87         }
88         item.keys = append(item.keys, k)
89         item.values = reflect.Append(item.values, v)
90         b.items[t] = item
91         return nil
92 }
93
94 func (b *appEngineBatch) Commit() error {
95         b.lock.Lock()
```

```
96          defer b.lock.Unlock()
97          var err error
98          for _, i := range b.items {
99                  if err == nil {
100                         _, err = datastore.PutMulti(b.c, i.keys,
                            ↪  i.values.Interface())
101                 }
102         }
103         b.items = make(map[reflect.Type]*appEngineBatchItem, 0)
104         return err
105 }
106
107 // appEngineStore is an key/value stored backed by AppEngine's datastore
108 type appEngineStore struct {
109         c context.Context
110 }
111
112 // Get returns an specified record by table and key
113 func (a appEngineStore) Get(table, key string, result interface{}) error
    ↪  {
114         k := datastore.NewKey(a.c, table, key, 0, nil)
115         err := datastore.Get(a.c, k, result)
116
117         if err == datastore.ErrNoSuchEntity {
118                 err = errKeyNotFound
119         }
120         if err == datastore.ErrInvalidEntityType {
121                 err = errResultMustBeAPointer
122         }
123         return newKeyError(table, key, err)
124 }
125
126 func (a appEngineStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
127         resultv := reflect.ValueOf(results)
128         resultt := resultv.Type()
129         err := make(map[string]error, len(keys))
130         if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
          ↪  reflect.String {
```

```
131                 for _, key := range keys {
132                         err[key] = errInvalidResults
133                 }
134                 return err
135         }
136         var elemt reflect.Type
137         isPtr := false
138         elemt = resultt.Elem()
139         if elemt.Kind() == reflect.Ptr {
140                 elemt = elemt.Elem()
141                 isPtr = true
142         }
143         appEngineKeys := make([]*datastore.Key, len(keys))
144         for i, key := range keys {
145                 appEngineKeys[i] = datastore.NewKey(a.c, table, key, 0,
                    ↪  nil)
146         }
147         resultReflection :=
        ↪  reflect.MakeSlice(reflect.SliceOf(reflect.PtrTo(elemt)),
        ↪  len(keys), len(keys))
148         resultTmp := resultReflection.Interface()
149         errGet := datastore.GetMulti(a.c, appEngineKeys, resultTmp)
150         multiErr, isMultiError := errGet.(appengine.MultiError)
151         if isMultiError {
152                 for i, key := range keys {
153                         err[key] = multiErr[i]
154                         if err[key] == datastore.ErrNoSuchEntity {
155                                 err[key] = newKeyError(table, key,
                                    ↪  errKeyNotFound)
156                         }
157                         if err[key] == nil {
158                                 keyv := reflect.ValueOf(key)
159                                 elemv := resultReflection.Index(i)
160                                 if !isPtr {
161                                         elemv = elemv.Elem()
162                                 }
163                                 resultv.SetMapIndex(keyv, elemv)
164                         }
165                 }
```

```
166         } else {
167                 for i, key := range keys {
168                         err[key] = errGet
169                         if err[key] == nil {
170                                 keyv := reflect.ValueOf(key)
171                                 elemv := resultReflection.Index(i)
172                                 if !isPtr {
173                                         elemv = elemv.Elem()
174                                 }
175                                 resultv.SetMapIndex(keyv, elemv)
176                         }
177                 }
178         }
179         return err
180 }
181
182 // Query returns a query manager to the specified table
183 func (a appEngineStore) Query(table string) Query {
184         return appEngineQuery{
185                 ctx: a.c,
186                 q:   datastore.NewQuery(table),
187         }
188 }
189
190 // GetAll returns an iterator that returns all the results in a table
191 func (a appEngineStore) GetAll(table string) Iterator {
192         return a.Query(table).Run()
193 }
194
195 // GetAllKeys returns an iterator that returns all the keys in a table
196 func (a appEngineStore) GetAllKeys(table string) Iterator {
197         return a.Query(table).KeysOnly().Run()
198 }
199
200 // Set puts/updates a key with a value in a Datastore's table
201 func (a appEngineStore) Set(table, key string, value interface{}) error {
202         k := datastore.NewKey(a.c, table, key, 0, nil)
203         _, e := datastore.Put(a.c, k, value)
204         if e == datastore.ErrInvalidEntityType {
```

```
205                 e = errValueMustBeAPointer
206         }
207         return e
208 }
209
210 // Del deletes the result from the datastore
211 func (a appEngineStore) Del(table, key string) error {
212         k := datastore.NewKey(a.c, table, key, 0, nil)
213         return datastore.Delete(a.c, k)
214 }
215
216 func (a appEngineStore) Batch() Batch {
217         items := make(map[reflect.Type]*appEngineBatchItem, 0)
218         return &appEngineBatch{
219                 c:     a.c,
220                 items: items,
221         }
222 }
223
224 func (a appEngineStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
225         // Deny access to non-referenced keys
226         return datastore.RunInTransaction(a.c, func(tc context.Context)
          ↪  error {
227                 return function(newOverlay(refs, NewAppEngineStore(tc)))
228         }, &datastore.TransactionOptions{
229                 XG: len(refs) > 1,
230         })
231 }
232
233 func (a appEngineStore) GenerateID(table string) (string, error) {
234         low, _, err := datastore.AllocateIDs(a.c, table, nil, 1)
235         var key string
236         if err == nil {
237                 key = fmt.Sprintf("%s-%d", table, low)
238         }
239         return key, err
240 }
241
```

```
242 // NewAppEngineStore returns an Key Value Store backed by Appengine's
    ↪  datastore
243 func NewAppEngineStore(c context.Context) Store {
244         return appEngineStore{c: c}
245 }
```

Arquivo badger.go

```
1  // +build !js
2
3  package kv
4
5  import (
6          "bytes"
7          "crypto/md5"
8          "fmt"
9          "math/rand"
10         "reflect"
11         "sync"
12         "time"
13
14         "github.com/dgraph-io/badger/v2"
15         "github.com/fjorgemota/gurudamatricula/util/clock"
16         "github.com/fjorgemota/gurudamatricula/util/coder"
17 )
18
19 func badgerTable(table string) []byte {
20         hash := md5.Sum([]byte(table))
21         return hash[:]
22 }
23
24 func badgerKey(table, key string) []byte {
25         buf := make([]byte, 0, 16+len(key)+1)
26         buf = append(buf, badgerTable(table)...)
27         buf = append(buf, ':')
28         buf = append(buf, key...)
29         return buf
30 }
31
32 // badgerIterator is an iterator for the badgerStore
```

```
33 type badgerIterator struct {
34         data   [][]byte
35         keys   []string
36         cod    coder.Coder
37         lock   sync.Mutex
38         cursor int
39         err    error
40 }
41
42 // Next returns the next result available in the iterator
43 func (i *badgerIterator) Next(result interface{}) (string, error) {
44         i.lock.Lock()
45         defer i.lock.Unlock()
46         err := i.err
47         var key string
48         resultv := reflect.ValueOf(result)
49         if resultv.Kind() == reflect.Ptr {
50                 resultv = resultv.Elem()
51         } else {
52                 err = errResultMustBeAPointer
53         }
54         if err == nil {
55                 if i.cursor >= len(i.keys) {
56                         resultv.Set(reflect.Zero(resultv.Type()))
57                         err = errDone
58                 }
59                 if err == nil {
60                         if i.data != nil {
61                                 err =
                                  ↪  i.cod.DecodeFrom(bytes.NewReader(i.data[i.curso
                                  ↪  result)
62                         }
63                         key = i.keys[i.cursor]
64                         i.cursor++
65                 }
66         }
67         return key, err
68 }
69
```

```
70 type badgerTransaction struct {
71         cod coder.Coder
72         tx  *badger.Txn
73         db  *badger.DB
74 }
75
76 // Get gets the specified record in the Bolt store, putting it into
   ↪ result
77 func (bt *badgerTransaction) Get(table, key string, result interface{})
   ↪ error {
78         var err error
79         if reflect.ValueOf(result).Kind() != reflect.Ptr {
80                 err = errResultMustBeAPointer
81         }
82         var item *badger.Item
83         if err == nil {
84                 item, err = bt.tx.Get(badgerKey(table, key))
85         }
86         if err == badger.ErrKeyNotFound {
87                 err = errKeyNotFound
88         }
89         if err == nil {
90                 err = item.Value(func(value []byte) error {
91                         return bt.cod.DecodeFrom(bytes.NewReader(value),
                           ↪ result)
92                 })
93         }
94         return newKeyError(table, key, err)
95 }
96
97 func (bt *badgerTransaction) Set(table, key string, result interface{})
   ↪ error {
98         var err error
99         resultv := reflect.ValueOf(result)
100        if resultv.Kind() == reflect.Ptr {
101                result = resultv.Elem().Interface()
102        } else {
103                err = errValueMustBeAPointer
104        }
```

```
105         var buf bytes.Buffer
106         if err == nil {
107                 err = bt.cod.EncodeTo(result, &buf)
108         }
109         if err == nil {
110                 err = bt.tx.Set(badgerKey(table, key), buf.Bytes())
111         }
112         return newKeyError(table, key, err)
113 }
114
115 func (bt *badgerTransaction) Del(table, key string) error {
116         return bt.tx.Delete(badgerKey(table, key))
117 }
118
119 func (bt *badgerTransaction) GenerateID(table string) (string, error) {
120         var id uint64
121         var err error
122         var sequence *badger.Sequence
123         if err == nil {
124                 sequence, err = bt.db.GetSequence([]byte(table), 1)
125         }
126         if err == nil {
127                 id, err = sequence.Next()
128         }
129         var result string
130         if err == nil {
131                 result = fmt.Sprintf("%s-%d", table, id)
132         }
133         return result, err
134 }
135
136 func (bt *badgerTransaction) Batch() Batch {
137         return newBatchKv(bt)
138 }
139
140 func (bt *badgerTransaction) GetAll(table string) Iterator {
141         var keys []string
142         var data [][]byte
143         var err error
```

```
144         prefix := append(badgerTable(table), ':')
145         it := bt.tx.NewIterator(badger.IteratorOptions{
146                 PrefetchValues: true,
147                 PrefetchSize:   100,
148                 Reverse:        false,
149                 AllVersions:    false,
150                 Prefix:         prefix,
151         })
152         defer it.Close()
153         for it.Seek(prefix); it.ValidForPrefix(prefix); it.Next() {
154                 item := it.Item()
155                 var value []byte
156                 if err == nil {
157                         value, err = item.ValueCopy(value)
158                 }
159                 if err == nil {
160                         keys = append(keys,
                        ↪  string(bytes.TrimPrefix(item.Key(), prefix)))
161                         data = append(data, value)
162                 }
163         }
164         return &badgerIterator{data: data, keys: keys, err: err, cod:
        ↪  bt.cod}
165 }
166
167 func (bt *badgerTransaction) GetAllKeys(table string) Iterator {
168         var keys []string
169         var err error
170         prefix := append(badgerTable(table), ':')
171         it := bt.tx.NewIterator(badger.IteratorOptions{
172                 PrefetchValues: false,
173                 PrefetchSize:   100,
174                 Reverse:        false,
175                 AllVersions:    false,
176         })
177         defer it.Close()
178         for it.Seek(prefix); it.ValidForPrefix(prefix); it.Next() {
179                 item := it.Item()
180                 if err == nil {
```

```
181                         keys = append(keys,
                            ↪  string(bytes.TrimPrefix(item.Key(), prefix)))
182                 }
183         }
184         return &badgerIterator{keys: keys, err: err}
185 }
186
187 func (bt *badgerTransaction) GetMulti(table string, keys []string,
    ↪  results interface{}) map[string]error {
188         return getMulti(bt, table, keys, results)
189 }
190
191 func newBadgerTransaction(db *badger.DB, tx *badger.Txn, cod coder.Coder)
    ↪  *badgerTransaction {
192         return &badgerTransaction{db: db, tx: tx, cod: cod}
193 }
194
195 type BadgerOptions struct {
196         Retries int
197         Clock   clock.Clock
198         Delay   time.Duration
199 }
200
201 // badgerStore is an store that saves data to Bolt
202 type badgerStore struct {
203         handle  *badger.DB
204         cod     coder.Coder
205         options BadgerOptions
206 }
207
208 // Get gets the specified record in the Bolt store, putting it into
    ↪  result
209 func (m badgerStore) Get(table, key string, result interface{}) error {
210         return m.retryView(func(tx *badger.Txn) error {
211                 return newBadgerTransaction(m.handle, tx,
                    ↪  m.cod).Get(table, key, result)
212         })
213 }
214
```

```
215 // GetAll returns an iterator with all the results found in the specified
    ↪  table
216 func (m badgerStore) GetAll(table string) Iterator {
217         var it *badgerIterator
218         m.retryView(func(tx *badger.Txn) error {
219                 it = newBadgerTransaction(m.handle, tx,
                    ↪  m.cod).GetAll(table).(*badgerIterator)
220                 return it.err
221         })
222         return it
223 }
224
225 // GetAllKeys returns an iterator with all the keys found in the
    ↪  specified table
226 func (m badgerStore) GetAllKeys(table string) Iterator {
227         var it *badgerIterator
228         m.retryView(func(tx *badger.Txn) error {
229                 it = newBadgerTransaction(m.handle, tx,
                    ↪  m.cod).GetAllKeys(table).(*badgerIterator)
230                 return it.err
231         })
232         return it
233 }
234
235 func (m badgerStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
236         var result map[string]error
237         m.retryView(func(tx *badger.Txn) error {
238                 result = newBadgerTransaction(m.handle, tx,
                    ↪  m.cod).GetMulti(table, keys, results)
239                 return nil
240         })
241         return result
242 }
243
244 // Query returns a query manager to the specified table
245 func (m badgerStore) Query(table string) Query {
246         return newQuery(m, table)
247 }
```

```
248
249 func (m badgerStore) retry(method func(func(txn *badger.Txn) error) error,
    ↪  fn func(txn *badger.Txn) error) error {
250         err := badger.ErrConflict
251         for c := 0; (err == badger.ErrConflict || err == badger.ErrRetry)
            ↪  && c < m.options.Retries; c++ {
252                 err = method(fn)
253                 if err != nil {
254                         m.options.Clock.Sleep(time.Duration((rand.Float64()
                        ↪  + 0.5) * float64(m.options.Delay)))
255                 }
256         }
257         return err
258 }
259
260 func (m badgerStore) retryUpdate(fn func(tx *badger.Txn) error) error {
261         return m.retry(m.handle.Update, fn)
262 }
263
264 func (m badgerStore) retryView(fn func(tx *badger.Txn) error) error {
265         return m.retry(m.handle.View, fn)
266 }
267
268 // Set puts the value on the table
269 func (m badgerStore) Set(table, key string, value interface{}) error {
270         return m.retryUpdate(func(tx *badger.Txn) error {
271                 return newBadgerTransaction(m.handle, tx,
                        ↪  m.cod).Set(table, key, value)
272         })
273 }
274
275 // Del deletes the key from the table
276 func (m badgerStore) Del(table, key string) error {
277         return m.retryUpdate(func(tx *badger.Txn) error {
278                 return newBadgerTransaction(m.handle, tx,
                        ↪  m.cod).Del(table, key)
279         })
280 }
281
```

```
282 func (m badgerStore) Batch() Batch {
283         return newBatchKv(m)
284 }
285
286 type badgerSetStore struct {
287         wb  *badger.WriteBatch
288         cod coder.Coder
289 }
290
291 func (bss badgerSetStore) Set(table, key string, result interface{})
    ↪  error {
292         var buf bytes.Buffer
293         err := bss.cod.EncodeTo(result, &buf)
294         if err == nil {
295                 err = bss.wb.Set(badgerKey(table, key), buf.Bytes())
296         }
297         return newKeyError(table, key, err)
298 }
299
300 func (m badgerStore) BatchCommit(function func(BatchSetStore) error)
    ↪  error {
301         wb := m.handle.NewWriteBatch()
302         defer wb.Cancel()
303         err := function(badgerSetStore{
304                 wb:  wb,
305                 cod: m.cod,
306         })
307         if err == nil {
308                 err = wb.Flush()
309         }
310         return err
311 }
312
313 func (m badgerStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
314         return m.retryUpdate(func(tx *badger.Txn) error {
315                 rollback :=
                    ↪  newRollbackStore(newBadgerTransaction(m.handle, tx,
                    ↪  m.cod))
```

```
316                 store := newOverlay(refs, rollback)
317                 err := function(store)
318                 if err == nil {
319                         err = rollback.commit()
320                 }
321                 return err
322         })
323 }
324
325 func (m badgerStore) GenerateID(table string) (string, error) {
326         var id string
327         err := m.retryUpdate(func(tx *badger.Txn) error {
328                 var errTransaction error
329                 id, errTransaction = newBadgerTransaction(m.handle, tx,
                    ↪  m.cod).GenerateID(table)
330                 return errTransaction
331         })
332         return id, err
333 }
334
335 // NewBadgerStore returns a new badger store
336 func NewBadgerStore(cod coder.Coder, handle *badger.DB) Store {
337         return NewBadgerStoreWithOptions(cod, handle, BadgerOptions{})
338 }
339
340 // NewBadgerStoreWithOptions returns a new badger store with additional
    ↪  options
341 func NewBadgerStoreWithOptions(cod coder.Coder, handle *badger.DB,
    ↪  options BadgerOptions) Store {
342         if options.Clock == nil {
343                 options.Clock = clock.NewRealClock()
344         }
345         if options.Retries < 1 {
346                 options.Retries = 3
347         }
348         if options.Delay <= 0 {
349                 options.Delay = 0
350         }
351         return &badgerStore{handle: handle, cod: cod, options: options}
```

```
352 }
```

Arquivo base.go

```
1  package kv
2
3  import "errors"
4
5  var (
6          // errKeyNotFound is returned when the key specified is not found
7          errKeyNotFound = errors.New("kv: key not found")
8          // errDone is returned when there are no more results to return
9          errDone = errors.New("kv: Query has no more results")
10         // errResultMustBeAPointer is returned when result is not a
           ↪  pointer..
11         errResultMustBeAPointer = errors.New("kv: result must be a
           ↪  pointer")
12         // errValueMustBeAPointer is returned when value is not a
           ↪  pointer..
13         errValueMustBeAPointer = errors.New("kv: value must be a
           ↪  pointer")
14         // errResultInvalidType is returned when result is not of the
           ↪  same type of variable
15         errResultInvalidType = errors.New("kv: The type on the table must
           ↪  be assignable to result")
16         // errAlreadyStarted is returned when KeysOnly() is called on a
           ↪  iterator already
17         // started
18         errAlreadyStarted = errors.New("kv: Iterator already started")
19
20         // errInvalidResults is returned when results is not a map in the
           ↪  expected
21         // format
22         errInvalidResults = errors.New("kv: results must be a map with
           ↪  key as a string")
23
24         // errAccessDenied is returned when the key mentioned is not
           ↪  referenced in the
25         // transaction
```

```
26          errAccessDenied = errors.New("kv: access denied for key not
            ↪  referenced in transaction")
27
28          // errOperationDenied is returned when the operation cannot be
            ↪  executed
29          // (like transactions inside transactions, for example)
30          errOperationDenied = errors.New("kv: access denied for that
            ↪  operation")
31  )
32
33  // Iterator allows the user to iterate by a larger dataset (e.g a full
    ↪  table)
34  // incrementally
35  type Iterator interface {
36          Next(result interface{}) (string, error)
37  }
38
39  // Query allows the user to query over the database to just return the
    ↪  results
40  // he wants
41  type Query interface {
42          Filter(field, comparator string, value interface{}) Query
43          KeysOnly() Query
44          Run() Iterator
45  }
46
47  // Batch allows the user to batch multiple set operations into a unique
48  // set of operations
49  type Batch interface {
50          Set(table, key string, value interface{}) error
51          Commit() error
52  }
53
54  // LimitedStore should be a key value store with basic capability and
    ↪  ease of
55  // use that is not queryable (no GetAll or Query methods)
56  type LimitedStore interface {
57          Get(table, key string, result interface{}) error
```

```
58          GetMulti(table string, keys []string, results interface{})
            ↪  map[string]error
59          Set(table, key string, value interface{}) error
60          Del(table, key string) error
61          Batch() Batch
62          GenerateID(table string) (string, error)
63  }
64
65  // Store should be a key value store with basic capability and ease of
    ↪  use
66  // that is queryable and can do things like get all the records of a
    ↪  table
67  // or do transactions
68  type Store interface {
69          LimitedStore
70          GetAll(table string) Iterator
71          GetAllKeys(table string) Iterator
72          Query(table string) Query
73          Transaction(refs []Reference, function func(LimitedStore) error)
            ↪  error
74  }
75
76  // Reference is a reference to a table and key in a key value store
77  type Reference struct {
78          Table string
79          Key   string
80  }
```

Arquivo batch.go

```
1   package kv
2
3   import (
4           "reflect"
5           "sync"
6   )
7
8   const batchItemLimit = 32
9
10  type batchItem struct {
```

```
        table string
        key   string
        value interface{}
}

// BatchSetStore is a store that can only set values
type BatchSetStore interface {
        Set(table, key string, value interface{}) error
}

type batchStore interface {
        BatchCommit(func(db BatchSetStore) error) error
}

type batchKv struct {
        buffer [batchItemLimit]batchItem
        count  int
        s      LimitedStore
        lock   sync.Mutex
        err    error
}

func (b *batchKv) Set(table, key string, value interface{}) error {
        b.lock.Lock()
        defer b.lock.Unlock()
        if b.err == nil && reflect.ValueOf(value).Kind() != reflect.Ptr {
                b.err = errValueMustBeAPointer
        }
        if b.err == nil {
                b.buffer[b.count] = batchItem{
                        table: table,
                        key:   key,
                        value: value,
                }
                b.count++
        }
        if b.count == batchItemLimit {
                b.commit()
        }
```

```
50          return b.err
51  }
52
53  func (b *batchKv) batchCommit(s BatchSetStore) error {
54          err := b.err
55          for err == nil && b.count > 0 {
56                  b.count--
57                  item := b.buffer[b.count]
58                  if err == nil {
59                          err = s.Set(item.table, item.key, item.value)
60                  }
61                  if err == nil {
62                          b.buffer[b.count].value = nil
63                  }
64          }
65          return err
66  }
67
68  func (b *batchKv) commit() error {
69          if b.err == nil {
70                  if s, ok := b.s.(batchStore); ok {
71                          b.err = s.BatchCommit(b.batchCommit)
72                  } else {
73                          b.err = b.batchCommit(b.s)
74                  }
75          }
76          return b.err
77  }
78
79  func (b *batchKv) Commit() error {
80          b.lock.Lock()
81          defer b.lock.Unlock()
82          err := b.commit()
83          return err
84  }
85
86  // newBatchKv returns an batcher that can batches set into memory before
87  // saving, one by one, into the kv.Store. Ideal for those stores who need
    ↪  to
```

```
88 // support Batch but cannot optimize it..
89 func newBatchKv(s LimitedStore) Batch {
90         return &batchKv{s: s}
91 }
```

Arquivo bbolt.go

```
1  // +build !js
2
3  package kv
4
5  import (
6          "bytes"
7          "fmt"
8          "reflect"
9          "sync"
10
11         "github.com/fjorgemota/gurudamatricula/util/pool"
12         "go.etcd.io/bbolt"
13
14         "github.com/fjorgemota/gurudamatricula/util/coder"
15 )
16
17 // bboltIterator is an iterator for the boltStore
18 type bboltIterator struct {
19         data   [][]byte
20         keys   []string
21         cod    coder.Coder
22         lock   sync.Mutex
23         cursor int
24         err    error
25 }
26
27 // Next returns the next result available in the iterator
28 func (i *bboltIterator) Next(result interface{}) (string, error) {
29         i.lock.Lock()
30         defer i.lock.Unlock()
31         err := i.err
32         var key string
33         resultv := reflect.ValueOf(result)
```

```
34          if resultv.Kind() == reflect.Ptr {
35                  resultv = resultv.Elem()
36          } else {
37                  err = errResultMustBeAPointer
38          }
39          if err == nil {
40                  if i.cursor >= len(i.keys) {
41                          resultv.Set(reflect.Zero(resultv.Type()))
42                          err = errDone
43                  }
44                  if err == nil {
45                          if i.data != nil {
46                                  reader :=
                                    ↪  bytes.NewReader(i.data[i.cursor])
47                                  err = i.cod.DecodeFrom(reader, result)
48                          }
49                          key = i.keys[i.cursor]
50                          i.cursor++
51                  }
52          }
53          return key, err
54  }
55
56  type bboltPut struct {
57          bucket *bbolt.Bucket
58          key    string
59          start  int
60          end    int
61  }
62
63  type bboltTransaction struct {
64          tx     *bbolt.Tx
65          buf    *bytes.Buffer
66          toPut []bboltPut
67          cod    coder.Coder
68  }
69
70  // Get gets the specified record in the BBolt store, putting it into
    ↪  result
```

```
func (m *bboltTransaction) Get(table, key string, result interface{})
↪  error {
        var err error
        if reflect.ValueOf(result).Kind() != reflect.Ptr {
                err = errResultMustBeAPointer
        }
        var bucket *bbolt.Bucket
        if err == nil {
                bucket = m.tx.Bucket([]byte(table))
                if bucket == nil {
                        err = errKeyNotFound
                }
        }
        var value []byte
        if err == nil {
                value = bucket.Get([]byte(key))
        }
        if value == nil && err == nil {
                err = errKeyNotFound
        } else if err == nil {
                err = m.cod.DecodeFrom(bytes.NewReader(value), result)
        }
        return newKeyError(table, key, err)
}

func (m *bboltTransaction) Set(table, key string, result interface{})
↪  error {
        var err error
        resultv := reflect.ValueOf(result)
        if resultv.Kind() == reflect.Ptr {
                result = resultv.Elem().Interface()
        } else {
                err = errValueMustBeAPointer
        }
        var bucket *bbolt.Bucket
        if err == nil {
                bucket = m.tx.Bucket([]byte(table))
                if bucket == nil {
                        bucket, err = m.tx.CreateBucket([]byte(table))
```

```
108                 }
109         }
110         start := m.buf.Len()
111         if err == nil {
112                 err = m.cod.EncodeTo(result, m.buf)
113         }
114         if err == nil {
115                 end := m.buf.Len()
116                 m.toPut = append(m.toPut, bboltPut{
117                         bucket: bucket,
118                         start:  start,
119                         end:    end,
120                         key:    key,
121                 })
122         }
123         return newKeyError(table, key, err)
124 }
125
126 func (m *bboltTransaction) Del(table, key string) error {
127         var err error
128         bucket := m.tx.Bucket([]byte(table))
129         if bucket != nil {
130                 err = bucket.Delete([]byte(key))
131         }
132         return err
133 }
134
135 func (m *bboltTransaction) GenerateID(table string) (string, error) {
136         bucket := m.tx.Bucket([]byte(table))
137         var err error
138         if bucket == nil {
139                 bucket, err = m.tx.CreateBucket([]byte(table))
140         }
141         var id uint64
142         if err == nil {
143                 id, err = bucket.NextSequence()
144         }
145         var result string
146         if err == nil {
```

```
147                 result = fmt.Sprintf("%s-%d", table, id)
148         }
149         return result, err
150 }
151
152 func (m *bboltTransaction) Batch() Batch {
153         return newBatchKv(m)
154 }
155 func (m *bboltTransaction) GetAll(table string) Iterator {
156         var keys []string
157         var data [][]byte
158         var err error
159         bucket := m.tx.Bucket([]byte(table))
160         if bucket != nil {
161                 err = bucket.ForEach(func(k []byte, v []byte) error {
162                         keys = append(keys, string(k))
163                         value := make([]byte, len(v))
164                         copy(value, v)
165                         data = append(data, value)
166                         return nil
167                 })
168         }
169         return &bboltIterator{data: data, keys: keys, err: err, cod:
          ↪  m.cod}
170 }
171
172 func (m *bboltTransaction) GetAllKeys(table string) Iterator {
173         var keys []string
174         var err error
175         bucket := m.tx.Bucket([]byte(table))
176         if bucket != nil {
177                 err = bucket.ForEach(func(k []byte, v []byte) error {
178                         keys = append(keys, string(k))
179                         return nil
180                 })
181         }
182         return &bboltIterator{keys: keys, err: err}
183 }
184
```

```
185 func (m *bboltTransaction) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
186         return getMulti(m, table, keys, results)
187 }
188
189 func (m *bboltTransaction) commitPut() error {
190         var err error
191         if m.buf != nil {
192                 bs := m.buf.Bytes()
193                 for _, put := range m.toPut {
194                         if err == nil {
195                                 err = put.bucket.Put([]byte(put.key),
                                    ↪  bs[put.start:put.end])
196                         }
197                 }
198         }
199         return err
200 }
201 func newBBoltTransaction(tx *bbolt.Tx, cod coder.Coder, put bool)
    ↪  *bboltTransaction {
202         var buf *bytes.Buffer
203         if tx.Writable() && put {
204                 buf = pool.GetBytesBuffer()
205         }
206         return &bboltTransaction{tx: tx, cod: cod, buf: buf}
207 }
208
209 // bboltStore is an store that saves data to Bolt
210 type bboltStore struct {
211         handle *bbolt.DB
212         cod    coder.Coder
213 }
214
215 // Get gets the specified record in the Bolt store, putting it into
    ↪  result
216 func (m *bboltStore) Get(table, key string, result interface{}) error {
217         return m.handle.View(func(tx *bbolt.Tx) error {
218                 return newBBoltTransaction(tx, m.cod, false).Get(table,
                    ↪  key, result)
```

```
219         })
220 }
221
222 // GetAll returns an iterator with all the results found in the specified
    ↪  table
223 func (m *bboltStore) GetAll(table string) Iterator {
224         var it *bboltIterator
225         m.handle.View(func(tx *bbolt.Tx) error {
226                 it = newBBoltTransaction(tx, m.cod,
                    ↪  false).GetAll(table).(*bboltIterator)
227                 return it.err
228         })
229         return it
230 }
231
232 // GetAllKeys returns an iterator with all the keys found in the
    ↪  specified table
233 func (m *bboltStore) GetAllKeys(table string) Iterator {
234         var it *bboltIterator
235         m.handle.View(func(tx *bbolt.Tx) error {
236                 it = newBBoltTransaction(tx, m.cod,
                    ↪  false).GetAllKeys(table).(*bboltIterator)
237                 return it.err
238         })
239         return it
240 }
241
242 func (m *bboltStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
243         var result map[string]error
244         m.handle.View(func(tx *bbolt.Tx) error {
245                 result = newBBoltTransaction(tx, m.cod,
                    ↪  false).GetMulti(table, keys, results)
246                 return nil
247         })
248         return result
249 }
250
251 // Query returns a query manager to the specified table
```

```
252 func (m *bboltStore) Query(table string) Query {
253         return newQuery(m, table)
254 }
255
256 // Set puts the value on the table
257 func (m *bboltStore) Set(table, key string, value interface{}) error {
258         var buf *bytes.Buffer
259         returnErr := m.handle.Batch(func(tx *bbolt.Tx) error {
260                 pool.PutBytesBuffer(buf)
261                 buf = nil
262                 transaction := newBBoltTransaction(tx, m.cod, true)
263                 err := transaction.Set(table, key, value)
264                 if err == nil {
265                         err = transaction.commitPut()
266                 }
267                 buf = transaction.buf
268                 return err
269         })
270         pool.PutBytesBuffer(buf)
271         return returnErr
272 }
273
274 // Del deletes the key from the table
275 func (m *bboltStore) Del(table, key string) error {
276         return m.handle.Batch(func(tx *bbolt.Tx) error {
277                 return newBBoltTransaction(tx, m.cod, false).Del(table,
                  ↪  key)
278         })
279 }
280
281 func (m *bboltStore) Batch() Batch {
282         return newBatchKv(m)
283 }
284
285 func (m *bboltStore) BatchCommit(function func(BatchSetStore) error)
    ↪  error {
286         var buf *bytes.Buffer
287         returnErr := m.handle.Batch(func(tx *bbolt.Tx) error {
288                 pool.PutBytesBuffer(buf)
```

```
289                 buf = nil
290                 transaction := newBBoltTransaction(tx, m.cod, true)
291                 err := function(transaction)
292                 if err == nil {
293                         err = transaction.commitPut()
294                 }
295                 buf = transaction.buf
296                 return err
297         })
298         pool.PutBytesBuffer(buf)
299         return returnErr
300 }
301
302 func (m *bboltStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
303         var buf *bytes.Buffer
304         returnErr := m.handle.Batch(func(tx *bbolt.Tx) error {
305                 pool.PutBytesBuffer(buf)
306                 buf = nil
307                 transaction := newBBoltTransaction(tx, m.cod, true)
308                 rollback := newRollbackStore(transaction)
309                 store := newOverlay(refs, rollback)
310                 err := function(store)
311                 if err == nil {
312                         err = rollback.commit()
313                 }
314                 if err == nil {
315                         err = transaction.commitPut()
316                 }
317                 buf = transaction.buf
318                 return err
319         })
320         pool.PutBytesBuffer(buf)
321         return returnErr
322 }
323
324 func (m *bboltStore) GenerateID(table string) (string, error) {
325         var id string
326         err := m.handle.Batch(func(tx *bbolt.Tx) error {
```

```
327                 var errTransaction error
328                 id, errTransaction = newBBoltTransaction(tx, m.cod,
                    ↪  false).GenerateID(table)
329                 return errTransaction
330         })
331         return id, err
332 }
333
334 // NewBBoltStore returns a new memory store
335 func NewBBoltStore(cod coder.Coder, handle *bbolt.DB) Store {
336         return &bboltStore{handle: handle, cod: cod}
337 }
```

Arquivo bolt.go

```
1  // +build !js
2
3  package kv
4
5  import (
6          "bytes"
7          "fmt"
8          "reflect"
9          "sync"
10
11         "github.com/boltdb/bolt"
12         "github.com/fjorgemota/gurudamatricula/util/pool"
13
14         "github.com/fjorgemota/gurudamatricula/util/coder"
15 )
16
17 // boltIterator is an iterator for the boltStore
18 type boltIterator struct {
19         data   [][]byte
20         keys   []string
21         cod    coder.Coder
22         lock   sync.Mutex
23         cursor int
24         err    error
25 }
```

```
26
27  // Next returns the next result available in the iterator
28  func (i *boltIterator) Next(result interface{}) (string, error) {
29          i.lock.Lock()
30          defer i.lock.Unlock()
31          err := i.err
32          var key string
33          resultv := reflect.ValueOf(result)
34          if resultv.Kind() == reflect.Ptr {
35                  resultv = resultv.Elem()
36          } else {
37                  err = errResultMustBeAPointer
38          }
39          if err == nil {
40                  if i.cursor >= len(i.keys) {
41                          resultv.Set(reflect.Zero(resultv.Type()))
42                          err = errDone
43                  }
44                  if err == nil {
45                          if i.data != nil {
46                                  reader :=
                                    ↪  bytes.NewReader(i.data[i.cursor])
47                                  err = i.cod.DecodeFrom(reader, result)
48                          }
49                          key = i.keys[i.cursor]
50                          i.cursor++
51                  }
52          }
53          return key, err
54  }
55
56  type boltPut struct {
57          bucket *bolt.Bucket
58          key    string
59          start  int
60          end    int
61  }
62
63  type boltTransaction struct {
```

```
        tx    *bolt.Tx
        buf   *bytes.Buffer
        toPut []boltPut
        cod   coder.Coder
}

// Get gets the specified record in the Bolt store, putting it into
↪ result
func (m *boltTransaction) Get(table, key string, result interface{})
↪ error {
        var err error
        if reflect.ValueOf(result).Kind() != reflect.Ptr {
                err = errResultMustBeAPointer
        }
        var bucket *bolt.Bucket
        if err == nil {
                bucket = m.tx.Bucket([]byte(table))
                if bucket == nil {
                        err = errKeyNotFound
                }
        }
        var value []byte
        if err == nil {
                value = bucket.Get([]byte(key))
        }
        if value == nil && err == nil {
                err = errKeyNotFound
        } else if err == nil {
                err = m.cod.DecodeFrom(bytes.NewReader(value), result)
        }
        return newKeyError(table, key, err)
}

func (m *boltTransaction) Set(table, key string, result interface{})
↪ error {
        var err error
        resultv := reflect.ValueOf(result)
        if resultv.Kind() == reflect.Ptr {
                result = resultv.Elem().Interface()
```

```
100         } else {
101                 err = errValueMustBeAPointer
102         }
103         var bucket *bolt.Bucket
104         if err == nil {
105                 bucket = m.tx.Bucket([]byte(table))
106                 if bucket == nil {
107                         bucket, err = m.tx.CreateBucket([]byte(table))
108                 }
109         }
110         start := m.buf.Len()
111         if err == nil {
112                 err = m.cod.EncodeTo(result, m.buf)
113         }
114         if err == nil {
115                 end := m.buf.Len()
116                 m.toPut = append(m.toPut, boltPut{
117                         bucket: bucket,
118                         start:  start,
119                         end:    end,
120                         key:    key,
121                 })
122         }
123         return newKeyError(table, key, err)
124 }
125
126 func (m *boltTransaction) Del(table, key string) error {
127         var err error
128         bucket := m.tx.Bucket([]byte(table))
129         if bucket != nil {
130                 err = bucket.Delete([]byte(key))
131         }
132         return err
133 }
134
135 func (m *boltTransaction) GenerateID(table string) (string, error) {
136         bucket := m.tx.Bucket([]byte(table))
137         var err error
138         if bucket == nil {
```

```
139                 bucket, err = m.tx.CreateBucket([]byte(table))
140         }
141         var id uint64
142         if err == nil {
143                 id, err = bucket.NextSequence()
144         }
145         var result string
146         if err == nil {
147                 result = fmt.Sprintf("%s-%d", table, id)
148         }
149         return result, err
150 }
151
152 func (m *boltTransaction) Batch() Batch {
153         return newBatchKv(m)
154 }
155 func (m *boltTransaction) GetAll(table string) Iterator {
156         var keys []string
157         var data [][]byte
158         var err error
159         bucket := m.tx.Bucket([]byte(table))
160         if bucket != nil {
161                 err = bucket.ForEach(func(k []byte, v []byte) error {
162                         keys = append(keys, string(k))
163                         value := make([]byte, len(v))
164                         copy(value, v)
165                         data = append(data, value)
166                         return nil
167                 })
168         }
169         return &boltIterator{data: data, keys: keys, err: err, cod:
          ↪  m.cod}
170 }
171
172 func (m *boltTransaction) GetAllKeys(table string) Iterator {
173         var keys []string
174         var err error
175         bucket := m.tx.Bucket([]byte(table))
176         if bucket != nil {
```

```
177                 err = bucket.ForEach(func(k []byte, v []byte) error {
178                         keys = append(keys, string(k))
179                         return nil
180                 })
181         }
182         return &boltIterator{keys: keys, err: err}
183 }
184
185 func (m *boltTransaction) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
186         return getMulti(m, table, keys, results)
187 }
188
189 func (m *boltTransaction) commitPut() error {
190         var err error
191         if m.buf != nil {
192                 bs := m.buf.Bytes()
193                 for _, put := range m.toPut {
194                         if err == nil {
195                                 err = put.bucket.Put([]byte(put.key),
                                    ↪  bs[put.start:put.end])
196                         }
197                 }
198         }
199         return err
200 }
201 func newBoltTransaction(tx *bolt.Tx, cod coder.Coder, put bool)
    ↪  *boltTransaction {
202         var buf *bytes.Buffer
203         if tx.Writable() && put {
204                 buf = pool.GetBytesBuffer()
205         }
206         return &boltTransaction{tx: tx, cod: cod, buf: buf}
207 }
208
209 // boltStore is an store that saves data to Bolt
210 type boltStore struct {
211         handle *bolt.DB
212         cod    coder.Coder
```

```
213 }
214
215 // Get gets the specified record in the Bolt store, putting it into
    ↪ result
216 func (m *boltStore) Get(table, key string, result interface{}) error {
217         return m.handle.View(func(tx *bolt.Tx) error {
218                 return newBoltTransaction(tx, m.cod, false).Get(table,
                    ↪ key, result)
219         })
220 }
221
222 // GetAll returns an iterator with all the results found in the specified
    ↪ table
223 func (m *boltStore) GetAll(table string) Iterator {
224         var it *boltIterator
225         m.handle.View(func(tx *bolt.Tx) error {
226                 it = newBoltTransaction(tx, m.cod,
                    ↪ false).GetAll(table).(*boltIterator)
227                 return it.err
228         })
229         return it
230 }
231
232 // GetAllKeys returns an iterator with all the keys found in the
    ↪ specified table
233 func (m *boltStore) GetAllKeys(table string) Iterator {
234         var it *boltIterator
235         m.handle.View(func(tx *bolt.Tx) error {
236                 it = newBoltTransaction(tx, m.cod,
                    ↪ false).GetAllKeys(table).(*boltIterator)
237                 return it.err
238         })
239         return it
240 }
241
242 func (m *boltStore) GetMulti(table string, keys []string, results
    ↪ interface{}) map[string]error {
243         var result map[string]error
244         m.handle.View(func(tx *bolt.Tx) error {
```

```
245                 result = newBoltTransaction(tx, m.cod,
                    ↪  false).GetMulti(table, keys, results)
246                 return nil
247         })
248         return result
249 }
250
251 // Query returns a query manager to the specified table
252 func (m *boltStore) Query(table string) Query {
253         return newQuery(m, table)
254 }
255
256 // Set puts the value on the table
257 func (m *boltStore) Set(table, key string, value interface{}) error {
258         var buf *bytes.Buffer
259         returnErr := m.handle.Batch(func(tx *bolt.Tx) error {
260                 pool.PutBytesBuffer(buf)
261                 buf = nil
262                 transaction := newBoltTransaction(tx, m.cod, true)
263                 err := transaction.Set(table, key, value)
264                 if err == nil {
265                         err = transaction.commitPut()
266                 }
267                 buf = transaction.buf
268                 return err
269         })
270         pool.PutBytesBuffer(buf)
271         return returnErr
272 }
273
274 // Del deletes the key from the table
275 func (m *boltStore) Del(table, key string) error {
276         return m.handle.Batch(func(tx *bolt.Tx) error {
277                 return newBoltTransaction(tx, m.cod, false).Del(table,
                    ↪  key)
278         })
279 }
280
281 func (m *boltStore) Batch() Batch {
```

```
282         return newBatchKv(m)
283 }
284
285 func (m *boltStore) BatchCommit(function func(BatchSetStore) error) error
    ↪ {
286         var buf *bytes.Buffer
287         returnErr := m.handle.Batch(func(tx *bolt.Tx) error {
288                 pool.PutBytesBuffer(buf)
289                 buf = nil
290                 transaction := newBoltTransaction(tx, m.cod, true)
291                 err := function(transaction)
292                 if err == nil {
293                         err = transaction.commitPut()
294                 }
295                 buf = transaction.buf
296                 return err
297         })
298         pool.PutBytesBuffer(buf)
299         return returnErr
300 }
301
302 func (m *boltStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
303         var buf *bytes.Buffer
304         returnErr := m.handle.Batch(func(tx *bolt.Tx) error {
305                 pool.PutBytesBuffer(buf)
306                 buf = nil
307                 transaction := newBoltTransaction(tx, m.cod, true)
308                 rollback := newRollbackStore(transaction)
309                 store := newOverlay(refs, rollback)
310                 err := function(store)
311                 if err == nil {
312                         err = rollback.commit()
313                 }
314                 if err == nil {
315                         err = transaction.commitPut()
316                 }
317                 buf = transaction.buf
318                 return err
```

```
319         })
320         pool.PutBytesBuffer(buf)
321         return returnErr
322 }
323
324 func (m *boltStore) GenerateID(table string) (string, error) {
325         var id string
326         err := m.handle.Batch(func(tx *bolt.Tx) error {
327                 var errTransaction error
328                 id, errTransaction = newBoltTransaction(tx, m.cod,
                    ↪  false).GenerateID(table)
329                 return errTransaction
330         })
331         return id, err
332 }
333
334 // NewBoltStore returns a new memory store
335 func NewBoltStore(cod coder.Coder, handle *bolt.DB) Store {
336         return &boltStore{handle: handle, cod: cod}
337 }
```

Arquivo cached.go

```
1  package kv
2
3  import (
4          "fmt"
5          "reflect"
6          "sync"
7
8          "github.com/fjorgemota/gurudamatricula/util/cache"
9  )
10
11 type cachedIterator struct {
12         it       Iterator
13         s        Store
14         keysOnly bool
15         table    string
16         lock     sync.Mutex
17 }
```

```
18
19 func (it *cachedIterator) Next(result interface{}) (string, error) {
20         it.lock.Lock()
21         defer it.lock.Unlock()
22         key, err := it.it.Next(result)
23         if !it.keysOnly && err == nil {
24                 err = it.s.Get(it.table, key, result)
25         }
26         return key, err
27 }
28
29 type cachedQuery struct {
30         q         Query
31         table     string
32         s         Store
33         keysOnly bool
34 }
35
36 func (q *cachedQuery) KeysOnly() Query {
37         return &cachedQuery{
38                 q:         q.q, // It is already a KeysOnly query..
39                 keysOnly: true,
40                 s:         q.s,
41                 table:     q.table,
42         }
43 }
44
45 func (q *cachedQuery) Filter(field, comparator string, value interface{})
   ↪  Query {
46         return &cachedQuery{
47                 q:         q.q.Filter(field, comparator, value),
48                 s:         q.s,
49                 keysOnly: q.keysOnly,
50                 table:     q.table,
51         }
52 }
53
54 func (q *cachedQuery) Run() Iterator {
55         return &cachedIterator{
```

```
                it:       q.q.Run(),
                s:        q.s,
                keysOnly: q.keysOnly,
                table:    q.table,
        }
}

type cachedBatch struct {
        b    Batch
        cac  cache.Cache
        buf  []batchItem
        lock sync.Mutex
        p    cachedStore
}

func (b *cachedBatch) Set(table, key string, value interface{}) error {
        b.lock.Lock()
        defer b.lock.Unlock()
        err := b.b.Set(table, key, value)
        if err == nil {
                b.buf = append(b.buf, batchItem{
                        table: table,
                        key:   key,
                        value: value,
                })
        }
        return err
}

func (b *cachedBatch) Commit() error {
        b.lock.Lock()
        defer b.lock.Unlock()
        err := b.b.Commit()
        for _, item := range b.buf {
                if err == nil {
                        err = b.cac.Set(b.p.formatKey(item.table,
                        ↪  item.key), item.value)
                }
        }
```

```
94          b.buf = nil
95          return err
96  }
97
98  type cachedStore struct {
99          kv  Store
100         cac cache.Cache
101 }
102
103 func (s cachedStore) formatKey(table, key string) string {
104         return fmt.Sprintf("%s-%s", table, key)
105 }
106
107 func (s cachedStore) Get(table, key string, result interface{}) error {
108         cachekey := s.formatKey(table, key)
109         err := s.cac.Get(cachekey, result)
110         if cache.IsGetError(err) {
111                 err = s.kv.Get(table, key, result)
112                 if err == nil {
113                         err = s.cac.Set(cachekey, result)
114                 }
115         }
116         return err
117 }
118
119 func (s cachedStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
120         resultv := reflect.ValueOf(results)
121         resultt := resultv.Type()
122         err := make(map[string]error, len(keys))
123         if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
          ↪  reflect.String {
124                 for _, key := range keys {
125                         err[key] = errInvalidResults
126                 }
127                 return err
128         }
129         cacheKeys := make([]string, len(keys))
130         for i, key := range keys {
```

```
131                 cacheKeys[i] = s.formatKey(table, key)
132         }
133         tmpCache := reflect.MakeMap(reflect.MapOf(resultt.Key(),
            ↪ resultt.Elem()))
134         errCache := s.cac.GetMulti(cacheKeys, tmpCache.Interface())
135         var errKeys []string
136         for i, key := range keys {
137                 cacheKey := cacheKeys[i]
138                 err[key] = errCache[cacheKey]
139                 if cache.IsGetError(err[key]) {
140                         errKeys = append(errKeys, key)
141                 } else {
142                         keyv := reflect.ValueOf(key)
143                         cacheKeyv := reflect.ValueOf(cacheKey)
144                         resultv.SetMapIndex(keyv,
                            ↪ tmpCache.MapIndex(cacheKeyv))
145                 }
146         }
147         if len(errKeys) > 0 {
148                 errGet := s.kv.GetMulti(table, errKeys, results)
149                 var sucKeys []string
150                 for _, key := range errKeys {
151                         err[key] = errGet[key]
152                         if err[key] == nil {
153                                 sucKeys = append(sucKeys, key)
154                         }
155                 }
156                 if len(sucKeys) > 0 {
157                         errSet := s.cac.SetMulti(sucKeys, results)
158                         for _, key := range sucKeys {
159                                 err[key] = errSet[key]
160                         }
161                 }
162         }
163         return err
164 }
165
166 // GetAll calls GetAll on the underlying store using KeysOnly mode
```

```
167 // And uses the keys to dispatch Get requests that may be cached
    ↪ automatically
168 // by this store
169 func (s cachedStore) GetAll(table string) Iterator {
170         q := s.kv.Query(table)
171         q = q.KeysOnly()
172         it := q.Run()
173         return &cachedIterator{
174                 s:     s,
175                 it:    it,
176                 table: table,
177         }
178 }
179
180 // GetAllKeys calls GetAllKeys on the underlying store and returns that
181 func (s cachedStore) GetAllKeys(table string) Iterator {
182         return s.kv.GetAllKeys(table)
183 }
184
185 // Query calls Query on the underlying store using KeysOnly mode
186 // And uses the keys to dispatch Get requests that may be cached
    ↪ automatically
187 // by this store
188 func (s cachedStore) Query(table string) Query {
189         q := s.kv.Query(table)
190         q = q.KeysOnly()
191         return &cachedQuery{
192                 q:     q,
193                 s:     s,
194                 table: table,
195         }
196 }
197 func (s cachedStore) Set(table, key string, value interface{}) error {
198         cachekey := s.formatKey(table, key)
199         nv := reflect.ValueOf(value)
200         if nv.Kind() == reflect.Ptr {
201                 nv = nv.Elem()
202         }
203         err := s.kv.Set(table, key, value)
```

```
204         if err == nil {
205                 err = s.cac.Set(cachekey, nv.Interface())
206         }
207         return err
208 }
209
210 func (s cachedStore) Del(table, key string) error {
211         err := s.kv.Del(table, key)
212         if err == nil {
213                 cachekey := s.formatKey(table, key)
214                 err = s.cac.Del(cachekey)
215         }
216         return err
217 }
218
219 func (s cachedStore) Batch() Batch {
220         return &cachedBatch{
221                 b:   s.kv.Batch(),
222                 cac: s.cac,
223                 p:   s,
224         }
225 }
226
227 func (s cachedStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
228         // Do not use CachedStore while wrapping the transaction store
            ↪  because
229         // we need to be consistent (other users maybe using the cache)
            ↪  and
230         // provide support for rollbacks
231         err := s.kv.Transaction(refs, function)
232         for _, ref := range refs {
233                 if err == nil {
234                         // Delete the keys that was referenced in the
                            ↪  transaction
235                         // from the cache so we do not have stale values
236                         err = s.cac.Del(s.formatKey(ref.Table, ref.Key))
237                 }
238         }
```

```
239         return err
240 }
241
242 func (s cachedStore) GenerateID(table string) (string, error) {
243         return s.kv.GenerateID(table)
244 }
245
246 // NewCachedStore returns a Store that is able to do caching
    ↪  automatically for
247 // some methods, like Set, Get and Del.
248 func NewCachedStore(kv Store, cac cache.Cache) Store {
249         return cachedStore{
250                 kv:  kv,
251                 cac: cac,
252         }
253 }
```

Arquivo datastore.go

```
1  // +build !js
2
3  package kv
4
5  import (
6          "fmt"
7          "reflect"
8          "sync"
9
10         "cloud.google.com/go/datastore"
11         "golang.org/x/net/context"
12         "google.golang.org/api/iterator"
13 )
14
15 func newDataStoreKey(table, key string) *datastore.Key {
16         return datastore.NameKey(table, key, nil)
17 }
18
19 func getDataStoreMulti(getMulti func(keys []*datastore.Key, result
   ↪  interface{}) error, table string, keys []string, results interface{})
   ↪  map[string]error {
```

```
20          resultv := reflect.ValueOf(results)
21          resultt := resultv.Type()
22          err := make(map[string]error, len(keys))
23          if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
            ↪  reflect.String {
24                  for _, key := range keys {
25                          err[key] = errInvalidResults
26                  }
27                  return err
28          }
29          var elemt reflect.Type
30          isPtr := false
31          elemt = resultt.Elem()
32          if elemt.Kind() == reflect.Ptr {
33                  elemt = elemt.Elem()
34                  isPtr = true
35          }
36          cloudDataKeys := make([]*datastore.Key, len(keys))
37          for i, key := range keys {
38                  cloudDataKeys[i] = newDataStoreKey(table, key)
39          }
40          resultReflection :=
            ↪  reflect.MakeSlice(reflect.SliceOf(reflect.PtrTo(elemt)),
            ↪  len(keys), len(keys))
41          resultTmp := resultReflection.Interface()
42          errGet := getMulti(cloudDataKeys, resultTmp)
43          multiErr, isMultiError := errGet.(datastore.MultiError)
44          if isMultiError {
45                  for i, key := range keys {
46                          err[key] = multiErr[i]
47                          if err[key] == datastore.ErrNoSuchEntity {
48                                  err[key] = newKeyError(table, key,
                                    ↪  errKeyNotFound)
49                          }
50                          if err[key] == nil {
51                                  keyv := reflect.ValueOf(key)
52                                  elemv := resultReflection.Index(i)
53                                  if !isPtr {
54                                          elemv = elemv.Elem()
```

```
55                                      }
56                                      resultv.SetMapIndex(keyv, elemv)
57                              }
58                      }
59              } else {
60                      for i, key := range keys {
61                              err[key] = errGet
62                              if err[key] == nil {
63                                      keyv := reflect.ValueOf(key)
64                                      elemv := resultReflection.Index(i)
65                                      if !isPtr {
66                                              elemv = elemv.Elem()
67                                      }
68                                      resultv.SetMapIndex(keyv, elemv)
69                              }
70                      }
71              }
72              return err
73      }
74
75      // cloudDataIterator is an iterator for DataStore queries
76      type cloudDataIterator struct {
77              it   *datastore.Iterator
78              lock sync.Mutex
79      }
80
81      // Next will return the next result available for the DataStore query
82      func (cdi *cloudDataIterator) Next(result interface{}) (string, error) {
83              cdi.lock.Lock()
84              defer cdi.lock.Unlock()
85              k, err := cdi.it.Next(result)
86              if err != nil {
87                      if err == iterator.Done {
88                              // Remap error from datastore.Done to ErrDone
89                              err = errDone
90                      }
91                      if err == datastore.ErrInvalidEntityType {
92                              err = errResultMustBeAPointer
93                      }
```

```
94                  return "", err
95          }
96          return k.Name, err
97  }
98
99  type cloudDataQuery struct {
100         query *datastore.Query
101         store cloudDataStore
102 }
103
104 func (cdq cloudDataQuery) Filter(field, comparator string, value
    ↪  interface{}) Query {
105         return cloudDataQuery{
106                 query: cdq.query.Filter(field+" "+comparator, value),
107                 store: cdq.store,
108         }
109 }
110
111 func (cdq cloudDataQuery) KeysOnly() Query {
112         return cloudDataQuery{
113                 query: cdq.query.KeysOnly(),
114                 store: cdq.store,
115         }
116 }
117
118 func (cdq cloudDataQuery) Run() Iterator {
119         return &cloudDataIterator{
120                 it: cdq.store.client.Run(cdq.store.ctx, cdq.query),
121         }
122 }
123
124 type cloudDataBatchItem struct {
125         keys   []*datastore.Key
126         values reflect.Value
127 }
128
129 type cloudDataBatch struct {
130         items    map[reflect.Type]*cloudDataBatchItem
131         putMulti func(keys []*datastore.Key, value interface{}) error
```

```
132         err      error
133         lock     sync.Mutex
134 }
135
136 func (cdb *cloudDataBatch) Set(table, key string, value interface{})
    ↪  error {
137         cdb.lock.Lock()
138         defer cdb.lock.Unlock()
139         k := newDataStoreKey(table, key)
140         v := reflect.ValueOf(value)
141         t := v.Type()
142         item, ok := cdb.items[t]
143         if ok && item.values.Len() > 50 {
144                 cdb.err = cdb.putMulti(item.keys,
                    ↪  item.values.Interface())
145                 ok = false
146         }
147         if !ok {
148                 item = &cloudDataBatchItem{
149                         values: reflect.MakeSlice(reflect.SliceOf(t), 0,
                            ↪  1),
150                 }
151         }
152         if cdb.err == nil {
153                 item.keys = append(item.keys, k)
154                 item.values = reflect.Append(item.values, v)
155                 cdb.items[t] = item
156         }
157         return cdb.err
158 }
159
160 func (cdb *cloudDataBatch) Commit() error {
161         cdb.lock.Lock()
162         defer cdb.lock.Unlock()
163         err := cdb.err
164         for _, i := range cdb.items {
165                 if err == nil {
166                         err = cdb.putMulti(i.keys, i.values.Interface())
167                 }
```

```
168         }
169         cdb.items = make(map[reflect.Type]*cloudDataBatchItem, 0)
170         return err
171 }
172
173 type cloudDataTransaction struct {
174         txn   *datastore.Transaction
175         store cloudDataStore
176 }
177
178 // Get gets the specified record in the Datastore, putting it into result
179 func (cdt cloudDataTransaction) Get(table, key string, result
    ↪  interface{}) error {
180         k := newDataStoreKey(table, key)
181         err := cdt.txn.Get(k, result)
182         if err == datastore.ErrNoSuchEntity {
183                 err = errKeyNotFound
184         }
185         if err == datastore.ErrInvalidEntityType {
186                 err = errResultMustBeAPointer
187         }
188         return newKeyError(table, key, err)
189 }
190
191 func (cdt cloudDataTransaction) Set(table, key string, value interface{})
    ↪  error {
192         k := newDataStoreKey(table, key)
193         _, err := cdt.txn.Put(k, value)
194         if err == datastore.ErrInvalidEntityType {
195                 err = errValueMustBeAPointer
196         }
197         return err
198 }
199
200 func (cdt cloudDataTransaction) Del(table, key string) error {
201         k := newDataStoreKey(table, key)
202         return cdt.txn.Delete(k)
203 }
204
```

```
func (cdt cloudDataTransaction) GenerateID(table string) (string, error) {
        return cdt.store.GenerateID(table)
}

func (cdt cloudDataTransaction) Batch() Batch {
        return &cloudDataBatch{
                putMulti: func(keys []*datastore.Key, value interface{}) error {
                        _, err := cdt.txn.PutMulti(keys, value)
                        return err
                },
                items: make(map[reflect.Type]*cloudDataBatchItem, 0),
        }
}

func (cdt cloudDataTransaction) GetMulti(table string, keys []string, results interface{}) map[string]error {
        return getDataStoreMulti(func(keys []*datastore.Key, result interface{}) error {
                return cdt.txn.GetMulti(keys, result)
        }, table, keys, results)
}

// cloudDataStore is an key/value stored backed by cloudData's datastore
type cloudDataStore struct {
        ctx    context.Context
        client *datastore.Client
}

// Get returns an specified record by table and key
func (cds cloudDataStore) Get(table, key string, result interface{}) error {
        k := newDataStoreKey(table, key)
        err := cds.client.Get(cds.ctx, k, result)
        if err == datastore.ErrNoSuchEntity {
                err = errKeyNotFound
        }
        if err == datastore.ErrInvalidEntityType {
```

```
239                 err = errResultMustBeAPointer
240         }
241         return newKeyError(table, key, err)
242 }
243
244 func (cds cloudDataStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
245         return getDataStoreMulti(func(keys []*datastore.Key, result
            ↪  interface{}) error {
246                 return cds.client.GetMulti(cds.ctx, keys, result)
247         }, table, keys, results)
248 }
249
250 // Query returns a query manager to the specified table
251 func (cds cloudDataStore) Query(table string) Query {
252         return cloudDataQuery{
253                 store: cds,
254                 query: datastore.NewQuery(table),
255         }
256 }
257
258 // GetAll returns an iterator that returns all the results in a table
259 func (cds cloudDataStore) GetAll(table string) Iterator {
260         return cds.Query(table).Run()
261 }
262
263 // GetAllkEYS returns an iterator that returns all the KEYS in a table
264 func (cds cloudDataStore) GetAllKeys(table string) Iterator {
265         return cds.Query(table).KeysOnly().Run()
266 }
267
268 // Set puts/updates a key with a value in a Datastore's table
269 func (cds cloudDataStore) Set(table, key string, value interface{}) error
    ↪ {
270         k := newDataStoreKey(table, key)
271         _, err := cds.client.Put(cds.ctx, k, value)
272         if err == datastore.ErrInvalidEntityType {
273                 err = errValueMustBeAPointer
274         }
```

```
275         return err
276 }
277
278 // Del deletes the result from the datastore
279 func (cds cloudDataStore) Del(table, key string) error {
280         k := newDataStoreKey(table, key)
281         return cds.client.Delete(cds.ctx, k)
282 }
283
284 func (cds cloudDataStore) Batch() Batch {
285         return &cloudDataBatch{
286                 putMulti: func(keys []*datastore.Key, value interface{})
                ↪  error {
287                         _, err := cds.client.PutMulti(cds.ctx, keys,
                        ↪  value)
288                         return err
289                 },
290                 items: make(map[reflect.Type]*cloudDataBatchItem, 0),
291         }
292 }
293
294 func (cds cloudDataStore) Transaction(refs []Reference, function
↪  func(LimitedStore) error) error {
295         // Deny access to non-referenced keys
296         _, err := cds.client.RunInTransaction(cds.ctx, func(txn
        ↪  *datastore.Transaction) error {
297                 return function(newOverlay(refs, cloudDataTransaction{
298                         store: cds,
299                         txn:   txn,
300                 }))
301         })
302         return err
303 }
304
305 func (cds cloudDataStore) GenerateID(table string) (string, error) {
306         keys := []*datastore.Key{
307                 {
308                         Kind: table,
309                 },
```

```
310         }
311         newKeys, err := cds.client.AllocateIDs(cds.ctx, keys)
312         var key string
313         if err == nil {
314                 key = fmt.Sprintf("%s-%d", table, newKeys[0].ID)
315         }
316         return key, err
317 }
318
319 // NewCloudDataStore returns an Key Value Store backed by Google Cloud
    ↪  DataStore
320 func NewCloudDataStore(client *datastore.Client, ctx context.Context)
    ↪  Store {
321         return cloudDataStore{
322                 ctx:    ctx,
323                 client: client,
324         }
325 }
```

Arquivo errors.go

```
1  package kv
2
3  import (
4          "fmt"
5
6          "github.com/pkg/errors"
7  )
8
9  type keyError struct {
10         key   string
11         table string
12         err   error
13 }
14
15 func (k keyError) Error() string {
16         return fmt.Sprintf("error on key '%s' on table '%s': %s", k.key,
           ↪  k.table, k.err.Error())
17 }
18
```

```
19 func (k keyError) Cause() error {
20         return k.err
21 }
22
23 func newKeyError(table, key string, err error) error {
24         if err == nil {
25                 return err
26         }
27         return errors.WithStack(keyError{err: err, key: key, table:
           ↪  table})
28 }
29 func IsKeyNotFoundError(err error) bool {
30         err = errors.Cause(err)
31         return err == errKeyNotFound
32 }
33
34 func IsResultMustBeAPointerError(err error) bool {
35         err = errors.Cause(err)
36         return err == errResultMustBeAPointer
37 }
38
39 func IsValueMustBeAPointer(err error) bool {
40         err = errors.Cause(err)
41         return err == errValueMustBeAPointer
42 }
43
44 func IsInvalidResultsError(err error) bool {
45         err = errors.Cause(err)
46         return err == errInvalidResults
47 }
48
49 func IsDoneError(err error) bool {
50         err = errors.Cause(err)
51         return err == errDone
52 }
53
54 func IsAccessDeniedError(err error) bool {
55         err = errors.Cause(err)
56         return err == errAccessDenied
```

```
57 }
58
59 func IsInvalidComparatorError(err error) bool {
60         err = errors.Cause(err)
61         return err == errInvalidComparator
62 }
63
64 func IsIncomparableError(err error) bool {
65         err = errors.Cause(err)
66         return err == errIncomparable
67 }
68
69 func IsFieldNotFoundError(err error) bool {
70         err = errors.Cause(err)
71         return err == errFieldNotFound
72 }
```

Arquivo file.go

```
1  package kv
2
3  import (
4          "crypto/md5"
5          "errors"
6          "fmt"
7          "reflect"
8          "strings"
9          "sync"
10
11         "github.com/fjorgemota/gurudamatricula/util/coder"
12         "github.com/fjorgemota/gurudamatricula/util/file"
13         uuid "github.com/gofrs/uuid"
14 )
15
16 // ErrInvalidTable is returned when the file processed does not
   ↪  correspond do
17 // the table expected
18 var ErrInvalidTable = errors.New("Invalid table")
19
20 type fileIterator struct {
```

```
21          it       file.Iterator
22          m        *fileStore
23          err      error
24          table    string
25          keysOnly bool
26          lock     sync.Mutex
27  }
28
29  func (it *fileIterator) Next(result interface{}) (string, error) {
30          it.lock.Lock()
31          defer it.lock.Unlock()
32          if it.err != nil {
33                  return "", it.err
34          }
35          err := errKeyNotFound
36          var key string
37          var filename string
38          for IsKeyNotFoundError(err) {
39                  filename, err = it.it.Next()
40                  if err == nil {
41                          // Yeah, for each file, we need to get a reader
                            ↪  for that file, read it,
42                          // decode, and return it to the user...
43                          key = it.m.getKey(it.table, filename)
44                          err = it.m.readInto(filename, result,
                            ↪  !it.keysOnly)
45                  }
46                  if file.IsDoneError(err) {
47                          err = errDone
48                  }
49          }
50          if err != nil {
51                  err = newKeyError(it.table, key, err)
52                  key = ""
53          }
54          return key, err
55  }
56
57  type fileStore struct {
```

```
58          m    file.Manager
59          cod  coder.Coder
60          lock sync.RWMutex
61  }
62
63  func (s *fileStore) encodeTable(table string) string {
64          return fmt.Sprintf("%x", md5.Sum([]byte(table)))
65  }
66  func (s *fileStore) getKey(table, file string) string {
67          hash := s.encodeTable(table)
68          result := strings.TrimPrefix(file, hash)
69          return result
70  }
71  func (s *fileStore) getFile(table, key string) string {
72          hash := s.encodeTable(table)
73          return hash + key
74  }
75  func (s *fileStore) readInto(filename string, result interface{}, decode
    ↪  bool) error {
76          reader, err := s.m.Reader(filename)
77          var value reflect.Value
78          if err == nil {
79                  value = reflect.ValueOf(result)
80                  if value.Kind() == reflect.Ptr {
81                          value = value.Elem()
82                  } else {
83                          err = errResultMustBeAPointer
84                  }
85          }
86          if err == nil && decode {
87                  err = s.cod.DecodeFrom(reader, result)
88          }
89          if err == nil {
90                  err = reader.Close()
91          }
92          if file.IsNotExistError(err) {
93                  err = errKeyNotFound
94          }
95          return err
```

```
96  }
97
98  func (s *fileStore) Get(table, key string, result interface{}) error {
99          s.lock.RLock()
100         defer s.lock.RUnlock()
101         filename := s.getFile(table, key)
102         return s.readInto(filename, result, true)
103 }
104
105 func (s *fileStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
106         return getMulti(s, table, keys, results)
107 }
108
109 func (s *fileStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
110         s.lock.Lock()
111         defer s.lock.Unlock()
112         // Please note that the command below will not work nice with
            ↪  shared
113         // file managers managed by multiple instances of this program
114         rollback := newRollbackStore(NewFileStore(s.m, s.cod))
115         store := newOverlay(refs, rollback)
116         err := function(store)
117         if err == nil {
118                 err = rollback.commit()
119         }
120         return err
121 }
122
123 func (s *fileStore) GetAll(table string) Iterator {
124         s.lock.RLock()
125         defer s.lock.RUnlock()
126         hash := s.encodeTable(table)
127         // Find files by prefix
128         it, err := s.m.Iterator(hash)
129         return &fileIterator{
130                 it:       it,
131                 table:    table,
```

```
132                 m:        s,
133                 err:      err,
134                 keysOnly: false,
135         }
136 }
137
138 func (s *fileStore) GetAllKeys(table string) Iterator {
139         s.lock.RLock()
140         defer s.lock.RUnlock()
141         hash := s.encodeTable(table)
142         // Find files by prefix
143         it, err := s.m.Iterator(hash)
144         return &fileIterator{
145                 it:       it,
146                 table:    table,
147                 m:        s,
148                 err:      err,
149                 keysOnly: true,
150         }
151 }
152
153 func (s *fileStore) Query(table string) Query {
154         // It's not efficient if it's a remote storage, but..
155         return newQuery(s, table)
156 }
157
158 func (s *fileStore) Set(table, key string, value interface{}) error {
159         s.lock.Lock()
160         defer s.lock.Unlock()
161         rv := reflect.ValueOf(value)
162         if rv.Kind() == reflect.Ptr {
163                 rv = rv.Elem()
164         } else {
165                 return errValueMustBeAPointer
166         }
167         value = rv.Interface()
168         file := s.getFile(table, key)
169         writer, err := s.m.Writer(file)
170         if err == nil {
```

```
171                 err = s.cod.EncodeTo(value, writer)
172         }
173         if err == nil {
174                 err = writer.Close()
175         }
176         return err
177 }
178
179 func (s *fileStore) Del(table, key string) error {
180         s.lock.Lock()
181         defer s.lock.Unlock()
182         filename := s.getFile(table, key)
183         err := s.m.Delete(filename)
184         if file.IsNotExistError(err) {
185                 err = nil
186         }
187         return err
188 }
189
190 func (s *fileStore) Batch() Batch {
191         return newBatchKv(s)
192 }
193
194 func (s *fileStore) GenerateID(table string) (string, error) {
195         var result string
196         id, err := uuid.NewV4()
197         if err == nil {
198                 result = table + "-" + id.String()
199         }
200         return result, nil
201 }
202
203 // NewFileStore returns a store that saves keys and values in files,
    ↪  managed by
204 // a file.Manager, and saved in local or remote storage
205 func NewFileStore(manager file.Manager, cod coder.Coder) Store {
206         return &fileStore{
207                 m:   manager,
208                 cod: cod,
```

```
209         }
210 }
```

Arquivo get_multi.go

```
1  package kv
2
3  import (
4          "reflect"
5  )
6
7  func getMulti(store LimitedStore, table string, keys []string, results
   ↪  interface{}) map[string]error {
8          resultv := reflect.ValueOf(results)
9          resultt := resultv.Type()
10         err := make(map[string]error, len(keys))
11         if resultt.Kind() != reflect.Map || resultt.Key().Kind() !=
           ↪  reflect.String {
12                 for _, key := range keys {
13                         err[key] = errInvalidResults
14                 }
15                 return err
16         }
17         var elemt reflect.Type
18         isPtr := false
19         elemt = resultt.Elem()
20         if elemt.Kind() == reflect.Ptr {
21                 elemt = elemt.Elem()
22                 isPtr = true
23         }
24         for _, key := range keys {
25                 elemv := reflect.New(elemt)
26                 v := elemv.Interface()
27                 err[key] = store.Get(table, key, v)
28                 if err[key] == nil {
29                         keyv := reflect.ValueOf(key)
30                         elemv = reflect.ValueOf(v)
31                         if !isPtr {
32                                 elemv = elemv.Elem()
33                         }
```

```
34                          resultv.SetMapIndex(keyv, elemv)
35                  }
36          }
37          return err
38  }
```

Arquivo iterator_q.go

```
1   package kv
2
3   import (
4           "errors"
5           "reflect"
6           "strings"
7           "sync"
8   )
9
10  var (
11          // errFieldNotFound is returned when the field specified is not
            ↪ found in struct
12          errFieldNotFound = errors.New("the field specified was not found
            ↪ in struct")
13          // errIncomparable is returned when the values cannot be compared
            ↪ by the filter
14          errIncomparable = errors.New("The types of the values are not
            ↪ comparable")
15          // errInvalidComparator is returned when the comparator is
            ↪ invalid
16          errInvalidComparator = errors.New("The comparator is invalid")
17  )
18
19  type comparatorType interface {
20          compare(rvalue reflect.Value, comparator string, rfvalue
            ↪ reflect.Value) (bool, error)
21  }
22
23  type comparatorInt struct{}
24
25  func (c *comparatorInt) compare(rvalue reflect.Value, comparator string,
    ↪ rfvalue reflect.Value) (bool, error) {
```

```
26          value := rvalue.Int()
27          fvalue := rfvalue.Int()
28          switch comparator {
29          case "<":
30                  return value < fvalue, nil
31          case "<=":
32                  return value <= fvalue, nil
33          case ">":
34                  return value > fvalue, nil
35          case ">=":
36                  return value >= fvalue, nil
37          }
38          return false, errInvalidComparator
39  }
40
41  type comparatorString struct{}
42
43  func (c *comparatorString) compare(rvalue reflect.Value, comparator
    ↪  string, rfvalue reflect.Value) (bool, error) {
44          value := rvalue.String()
45          fvalue := rfvalue.String()
46          switch comparator {
47          case "<":
48                  return value < fvalue, nil
49          case "<=":
50                  return value <= fvalue, nil
51          case ">":
52                  return value > fvalue, nil
53          case ">=":
54                  return value >= fvalue, nil
55          }
56          return false, errInvalidComparator
57  }
58
59  type comparatorFloat struct{}
60
61  func (c *comparatorFloat) compare(rvalue reflect.Value, comparator string,
    ↪  rfvalue reflect.Value) (bool, error) {
62          value := rvalue.Float()
```

```
63          fvalue := rfvalue.Float()
64          switch comparator {
65          case "<":
66                  return value < fvalue, nil
67          case "<=":
68                  return value <= fvalue, nil
69          case ">":
70                  return value > fvalue, nil
71          case ">=":
72                  return value >= fvalue, nil
73          }
74          return false, errInvalidComparator
75  }
76
77  type comparatorUint struct{}
78
79  func (c *comparatorUint) compare(rvalue reflect.Value, comparator string,
    ↪  rfvalue reflect.Value) (bool, error) {
80          value := rvalue.Uint()
81          fvalue := rfvalue.Uint()
82          switch comparator {
83          case "<":
84                  return value < fvalue, nil
85          case "<=":
86                  return value <= fvalue, nil
87          case ">":
88                  return value > fvalue, nil
89          case ">=":
90                  return value >= fvalue, nil
91          }
92          return false, errInvalidComparator
93  }
94
95  func detectKind(k reflect.Kind) reflect.Kind {
96          switch k {
97          case reflect.Int, reflect.Int8, reflect.Int16, reflect.Int32,
            ↪  reflect.Int64:
98                  return reflect.Int
```

```
99          case reflect.Uint, reflect.Uintptr, reflect.Uint8, reflect.Uint16,
            ↪  reflect.Uint32, reflect.Uint64:
100                 return reflect.Uint
101         case reflect.Float32, reflect.Float64:
102                 return reflect.Float32
103         case reflect.String:
104                 return reflect.String
105         default:
106                 return reflect.Invalid
107         }
108 }
109 func detectComparator(rvalue reflect.Value, rfvalue reflect.Value)
    ↪  (comparatorType, error) {
110         krvalue := detectKind(rvalue.Kind())
111         krfvalue := detectKind(rfvalue.Kind())
112         if krvalue != krfvalue {
113                 return nil, errIncomparable
114         }
115         switch krvalue {
116         case reflect.Int:
117                 return &comparatorInt{}, nil
118         case reflect.Float32:
119                 return &comparatorFloat{}, nil
120         case reflect.Uint:
121                 return &comparatorUint{}, nil
122         case reflect.String:
123                 return &comparatorString{}, nil
124         default:
125                 return nil, errIncomparable
126         }
127 }
128
129 type filter struct {
130         field, ft, comparator string
131         ind                   int
132         t                     reflect.Type
133         value                 interface{}
134         rvalue                reflect.Value
135 }
```

```
136
137 func (f *filter) compare(key string, value interface{}) (bool, error) {
138         rvalue := reflect.ValueOf(value)
139         rfvalue := f.rvalue
140         if !rvalue.Type().Comparable() || !rfvalue.Type().Comparable() {
141                 return false, errIncomparable
142         }
143         if f.comparator == "=" {
144                 // Short circuit if the comparator is a simple one
145                 return value == f.value, nil
146         }
147         comp, err := detectComparator(rvalue, rfvalue)
148         if err != nil {
149                 return false, err
150         }
151         return comp.compare(rvalue, f.comparator, rfvalue)
152 }
153
154 func (f *filter) findField(resultt reflect.Type) error {
155         if f.ind >= 0 && resultt == f.t || f.field == "__key__" {
156                 return nil
157         }
158         f.t = resultt
159         nf := resultt.NumField()
160         var ind int
161         var found bool
162         for ind = 0; ind < nf; ind++ {
163                 field := f.t.Field(ind)
164                 if field.Tag.Get("db") == f.field || field.Name == f.ft {
165                         found = true
166                         break
167                 }
168         }
169         f.ind = ind
170         if !found {
171                 return errFieldNotFound
172         }
173         return nil
174
```

```
}
func (f *filter) Compare(key string, resultv reflect.Value) (bool, error)
↪ {
        resultt := resultv.Type()
        err := f.findField(resultt)
        if err != nil {
                return false, err
        }
        var v interface{}
        if f.field == "__key__" {
                v = key
        } else {
                v = resultv.Field(f.ind).Interface()
        }
        return f.compare(key, v)
}

// iteratorQuery is a iterator that filter results based on the equality
↪  between
// a field in each record with it's value
type iteratorQuery struct {
        parent   Iterator
        err      error
        keysOnly bool
        filters  []*filter
        lock     sync.Mutex
}

// Next returns the next result in the iterator, filtering by the field
// specified
func (i *iteratorQuery) Next(result interface{}) (string, error) {
        i.lock.Lock()
        defer i.lock.Unlock()
        resultv := reflect.ValueOf(result)
        var resultt reflect.Type
        if resultv.Kind() == reflect.Ptr {
                resultv = resultv.Elem()
                resultt = resultv.Type()
        } else {
```

```
212                 return "", errResultMustBeAPointer
213         }
214         for {
215                 resultn := reflect.New(resultt)
216                 key, err := i.parent.Next(resultn.Interface())
217                 if err != nil {
218                         return "", err
219                 }
220                 resultn = resultn.Elem()
221                 match := true
222                 for _, filter := range i.filters {
223                         matchFilter, err := filter.Compare(key, resultn)
224                         if err != nil {
225                                 return "", err
226                         }
227                         if !matchFilter {
228                                 match = false
229                                 break
230                         }
231                 }
232                 if match {
233                         if !i.keysOnly {
234                                 resultv.Set(resultn)
235                         }
236                         return key, nil
237                 }
238         }
239 }
240
241 type query struct {
242         store    Store
243         table    string
244         keysOnly bool
245         filters  []*filter
246 }
247
248 func (q *query) KeysOnly() Query {
249         filters := make([]*filter, len(q.filters))
250         copy(filters, q.filters)
```

```
251         return &query{
252                 store:    q.store,
253                 table:    q.table,
254                 keysOnly: true,
255                 filters:  filters,
256         }
257 }
258
259 func (q *query) Filter(field, comparator string, value interface{}) Query
    ↪ {
260         ft := strings.Replace(field, "_", " ", -1)
261         ft = strings.Title(ft)
262         ft = strings.Replace(ft, " ", "", -1)
263         filters := make([]*filter, len(q.filters))
264         copy(filters, q.filters)
265         filters = append(filters, &filter{
266                 field:      field,
267                 comparator: comparator,
268                 value:      value,
269                 ft:         ft,
270                 rvalue:     reflect.ValueOf(value),
271                 ind:        -1,
272         })
273         return &query{
274                 store:    q.store,
275                 table:    q.table,
276                 keysOnly: q.keysOnly,
277                 filters:  filters,
278         }
279 }
280
281 func (q *query) Run() Iterator {
282         var err error
283         var it Iterator
284         if q.keysOnly && len(q.filters) == 0 {
285                 it = q.store.GetAllKeys(q.table)
286         } else {
287                 it = q.store.GetAll(q.table)
288         }
```

```
289         return &iteratorQuery{
290                 keysOnly: q.keysOnly,
291                 filters:  q.filters,
292                 parent:   it,
293                 err:      err,
294         }
295 }
296
297 // NewQuery returns an iterator that receives a parent iterator, a field
298 // and a value, and returns only values that matches the field and value
299 // specified
300 func newQuery(store Store, table string) Query {
301         var filters []*filter
302         return &query{
303                 store:   store,
304                 table:   table,
305                 filters: filters,
306         }
307 }
```

Arquivo logger.go

```
1  package kv
2
3  import (
4          uuid "github.com/gofrs/uuid"
5          "github.com/sirupsen/logrus"
6  )
7
8  type loggerIterator struct {
9          it     Iterator
10         logger logrus.FieldLogger
11 }
12
13 func (it *loggerIterator) Next(result interface{}) (string, error) {
14         id, err := it.it.Next(result)
15         it.logger.WithError(err).WithField("id", id).Debug("Returning
           ↪  result")
16         return id, err
17 }
```

```
18
19 type loggerQuery struct {
20         q      Query
21         logger logrus.FieldLogger
22 }
23
24 func (q *loggerQuery) KeysOnly() Query {
25         return &loggerQuery{
26                 q:      q.q.KeysOnly(), // It is already a KeysOnly
                   ↪  query..
27                 logger: q.logger.WithField("keys_only", true),
28         }
29 }
30
31 func (q *loggerQuery) Filter(field, comparator string, value interface{})
   ↪  Query {
32         return &loggerQuery{
33                 q:      q.q.Filter(field, comparator, value),
34                 logger: q.logger.WithField("field_"+field+comparator,
                   ↪  value),
35         }
36 }
37
38 func (q *loggerQuery) Run() Iterator {
39         q.logger.Debug("Running query")
40         return &loggerIterator{
41                 it:     q.q.Run(),
42                 logger: q.logger,
43         }
44 }
45
46 type loggerBatch struct {
47         b      Batch
48         logger logrus.FieldLogger
49         err    error
50 }
51
52 func (b *loggerBatch) Set(table, key string, value interface{}) error {
53         localErr := b.err
```

```
54          if localErr == nil {
55                  localErr = b.b.Set(table, key, value)
56                  b.logger.WithError(localErr).WithFields(logrus.Fields{"table":
                    ↪  table, "key": key}).Debug("Set key")
57          }
58          return localErr
59  }
60
61  func (b *loggerBatch) Commit() error {
62          localErr := b.err
63          if localErr == nil {
64                  localErr = b.b.Commit()
65                  b.logger.WithError(localErr).Debug("Commiting batch")
66          }
67          return localErr
68  }
69
70  type loggerTransaction struct {
71          kv     LimitedStore
72          logger logrus.FieldLogger
73  }
74
75  func (s loggerTransaction) Get(table, key string, result interface{})
    ↪  error {
76          err := s.kv.Get(table, key, result)
77          s.logger.WithError(err).WithFields(logrus.Fields{"table": table,
            ↪  "key": key}).Debug("Getting key")
78          return err
79  }
80  func (s loggerTransaction) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
81          result := s.kv.GetMulti(table, keys, results)
82          s.logger.WithFields(logrus.Fields{"table": table, "keys": keys,
            ↪  "result": result}).Debug("Getting multiple keys")
83          return result
84  }
85  func (s loggerTransaction) Set(table, key string, value interface{})
    ↪  error {
86          err := s.kv.Set(table, key, value)
```

```
87          s.logger.WithError(err).WithFields(logrus.Fields{"table": table,
            ↪  "key": key}).Debug("Setting key")
88          return err
89  }
90  func (s loggerTransaction) Del(table, key string) error {
91          err := s.kv.Del(table, key)
92          s.logger.WithError(err).WithFields(logrus.Fields{"table": table,
            ↪  "key": key}).Debug("Delete key")
93          return err
94  }
95  func (s loggerTransaction) Batch() Batch {
96          id, err := uuid.NewV4()
97          var logger logrus.FieldLogger
98          if err == nil {
99                  logger = s.logger.WithField("batch_id", id.String())
100                 logger.Debug("Starting batch..")
101         }
102         return &loggerBatch{
103                 b:      s.kv.Batch(),
104                 logger: logger,
105                 err:    err,
106         }
107 }
108 func (s loggerTransaction) GenerateID(table string) (string, error) {
109         id, err := s.kv.GenerateID(table)
110         s.logger.WithError(err).WithFields(logrus.Fields{"table": table,
            ↪  "id": id}).Debug("Generating ID")
111         return id, err
112 }
113
114 func newLoggerTransaction(kv LimitedStore, logger logrus.FieldLogger)
    ↪  LimitedStore {
115         return loggerTransaction{kv: kv, logger: logger}
116 }
117
118 type loggerStore struct {
119         kv     Store
120         logger logrus.FieldLogger
121 }
```

```
122
123 func (s loggerStore) Get(table, key string, result interface{}) error {
124         return newLoggerTransaction(s.kv, s.logger).Get(table, key,
        ↪  result)
125 }
126
127 func (s loggerStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
128         return newLoggerTransaction(s.kv, s.logger).GetMulti(table, keys,
        ↪  results)
129 }
130
131 // GetAll calls GetAll on the underlying store using KeysOnly mode
132 // And uses the keys to dispatch Get requests that may be logger
    ↪  automatically
133 // by this store
134 func (s loggerStore) GetAll(table string) Iterator {
135         return &loggerIterator{
136                 it:     s.kv.GetAll(table),
137                 logger: s.logger,
138         }
139 }
140
141 // GetAllKeys calls GetAllKeys on the underlying store using KeysOnly
    ↪  mode
142 // And uses the keys to dispatch Get requests that will be logged
    ↪  automatically
143 // by this store
144 func (s loggerStore) GetAllKeys(table string) Iterator {
145         return &loggerIterator{
146                 it:     s.kv.GetAllKeys(table),
147                 logger: s.logger,
148         }
149 }
150
151 // Query calls Query on the underlying store using KeysOnly mode
152 // And uses the keys to dispatch Get requests that may be logger
    ↪  automatically
153 // by this store
```

```
154 func (s loggerStore) Query(table string) Query {
155         return &loggerQuery{
156                 q:      s.kv.Query(table),
157                 logger: s.logger,
158         }
159 }
160 func (s loggerStore) Set(table, key string, value interface{}) error {
161         return newLoggerTransaction(s.kv, s.logger).Set(table, key,
        ↪  value)
162 }
163
164 func (s loggerStore) Del(table, key string) error {
165         return newLoggerTransaction(s.kv, s.logger).Del(table, key)
166 }
167
168 func (s loggerStore) Batch() Batch {
169         return newLoggerTransaction(s.kv, s.logger).Batch()
170 }
171
172 func (s loggerStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
173         logger := s.logger.WithField("transaction_id",
        ↪  uuid.Must(uuid.NewV4()).String())
174         err := s.kv.Transaction(refs, func(db LimitedStore) error {
175                 err := function(newLoggerTransaction(db, logger))
176                 logger.WithError(err).Debug("Preparing to
                ↪  commit/rollback")
177                 return err
178         })
179         logger.WithError(err).Debug("Transaction finished")
180         return err
181 }
182
183 func (s loggerStore) GenerateID(table string) (string, error) {
184         id, err := s.kv.GenerateID(table)
185         s.logger.WithError(err).WithFields(logrus.Fields{"table": table,
        ↪  "id": id}).Debug("Generating ID")
186         return id, err
187 }
```

```
188
189 // NewLoggerStore returns a Store that is able to do caching
    ↪  automatically for
190 // some methods, like Set, Get and Del.
191 func NewLoggerStore(kv Store, logger logrus.FieldLogger) Store {
192         return loggerStore{
193                 kv:     kv,
194                 logger: logger.WithField("store_id",
                    ↪  uuid.Must(uuid.NewV4()).String()),
195         }
196 }
```

Arquivo memory.go

```
1  package kv
2
3  import (
4          "bytes"
5          "reflect"
6          "sync"
7
8          "github.com/fjorgemota/gurudamatricula/util/coder"
9          uuid "github.com/gofrs/uuid"
10 )
11
12 // memoryIterator is an iterator for the memoryStore
13 type memoryIterator struct {
14         coder  coder.Coder
15         data   [][]byte
16         keys   []string
17         lock   sync.Mutex
18         cursor int
19         err    error
20 }
21
22 // Next returns the next result available in the iterator
23 func (i *memoryIterator) Next(result interface{}) (string, error) {
24         i.lock.Lock()
25         defer i.lock.Unlock()
26         var key string
```

```
27          if i.err == nil {
28                  resultv := reflect.ValueOf(result)
29                  if resultv.Kind() != reflect.Ptr {
30                          i.err = errResultMustBeAPointer
31                  }
32                  if i.err == nil && i.cursor >= len(i.keys) {
33                          i.err = errDone
34                  }
35                  if i.err == nil {
36                          if i.data != nil {
37                                  i.err =
                                    ↪  i.coder.DecodeFrom(bytes.NewReader(i.data[i.cur
                                    ↪  result)
38                          }
39                          key = i.keys[i.cursor]
40                          i.cursor++
41                  }
42          }
43          return key, i.err
44  }
45
46  // memoryStore is an store without any persistence, it only works in
    ↪  memory
47  type memoryStore struct {
48          lock  sync.RWMutex
49          data  map[string]map[string][]byte
50          coder coder.Coder
51  }
52
53  // Get gets the specified record in the memory, putting it into result
54  func (m *memoryStore) Get(table, key string, result interface{}) error {
55          m.lock.RLock()
56          defer m.lock.RUnlock()
57          tab, ok := m.data[table]
58          if !ok {
59                  return newKeyError(table, key, errKeyNotFound)
60          }
61          var source []byte
62          source, ok = tab[key]
```

```
63          if !ok {
64                  return newKeyError(table, key, errKeyNotFound)
65          }
66          var err error
67          if reflect.ValueOf(result).Kind() == reflect.Ptr {
68                  reader := bytes.NewReader(source)
69                  err = m.coder.DecodeFrom(reader, result)
70          } else {
71                  err = errResultMustBeAPointer
72          }
73          return err
74  }
75
76  // GetAll returns an iterator with all the results found in the specified
    ↪  table
77  func (m *memoryStore) GetAll(table string) Iterator {
78          m.lock.RLock()
79          defer m.lock.RUnlock()
80          tab, ok := m.data[table]
81          if !ok {
82                  return &memoryIterator{err: errDone}
83          }
84          data := make([][]byte, len(tab))
85          keys := make([]string, len(tab))
86          i := 0
87          for k, v := range tab {
88                  data[i] = make([]byte, len(v))
89                  copy(data[i], v)
90                  keys[i] = k
91                  i++
92          }
93          return &memoryIterator{data: data, keys: keys, coder: m.coder}
94  }
95
96  // GetAllKeys returns an iterator with all the keys found in the
    ↪  specified table
97  func (m *memoryStore) GetAllKeys(table string) Iterator {
98          m.lock.RLock()
99          defer m.lock.RUnlock()
```

```
100         tab, ok := m.data[table]
101         if !ok {
102                 return &memoryIterator{err: errDone}
103         }
104         keys := make([]string, len(tab))
105         i := 0
106         for k := range tab {
107                 keys[i] = k
108                 i++
109         }
110         return &memoryIterator{keys: keys}
111 }
112
113 func (m *memoryStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
114         return getMulti(m, table, keys, results)
115 }
116
117 // Query returns a query manager to the specified table
118 func (m *memoryStore) Query(table string) Query {
119         return newQuery(m, table)
120 }
121
122 // Set puts the value on the table
123 func (m *memoryStore) Set(table, key string, value interface{}) error {
124         m.lock.Lock()
125         defer m.lock.Unlock()
126         valuev := reflect.ValueOf(value)
127         if valuev.Kind() != reflect.Ptr {
128                 return errValueMustBeAPointer
129         }
130         _, ok := m.data[table]
131         if !ok {
132                 m.data[table] = make(map[string][]byte, 1)
133         }
134         var buf bytes.Buffer
135         err := m.coder.EncodeTo(value, &buf)
136         if err == nil {
137                 m.data[table][key] = buf.Bytes()
```

```
138         }
139         return err
140 }
141
142 // Del deletes the key from the table
143 func (m *memoryStore) Del(table, key string) error {
144         m.lock.Lock()
145         defer m.lock.Unlock()
146         tab, ok := m.data[table]
147         if !ok {
148                 // Well, the table does not exist, but the key also do
                    ↪  not exists
149                 // So, we can consider it as deleted. :P
150                 return nil
151         }
152         // Well, the table does not exist, but the key also do not exists
153         // So, we can consider it as deleted. :P
154         delete(tab, key)
155         return nil
156 }
157
158 func (m *memoryStore) Batch() Batch {
159         return newBatchKv(m)
160 }
161
162 func (m *memoryStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
163         m.lock.Lock()
164         defer m.lock.Unlock()
165         rollback := newRollbackStore(&memoryStore{data: m.data, coder:
            ↪  m.coder})
166         store := newOverlay(refs, rollback)
167         err := function(store)
168         if err == nil {
169                 err = rollback.commit()
170         }
171         return err
172 }
173
```

```
174 func (m *memoryStore) GenerateID(table string) (string, error) {
175         var result string
176         id, err := uuid.NewV4()
177         if err == nil {
178                 result = table + "-" + id.String()
179         }
180         return result, nil
181 }
182
183 // NewMemoryStore returns a new memory store
184 func NewMemoryStore() Store {
185         return NewMemoryStoreWithCoder(coder.NewGobCoder())
186 }
187
188 func NewMemoryStoreWithCoder(coder coder.Coder) Store {
189         data := make(map[string]map[string][]byte, 0)
190         return &memoryStore{data: data, coder: coder}
191 }
```

Arquivo multiplex.go

```
1  package kv
2
3  import (
4          "bytes"
5
6          "github.com/fjorgemota/gurudamatricula/util/coder"
7          "github.com/fjorgemota/gurudamatricula/util/pool"
8  )
9
10 type MultiplexItem struct {
11         Value []byte
12 }
13 type multiplexIterator struct {
14         err error
15         it  Iterator
16 }
17
18 func (it *multiplexIterator) Next(result interface{}) (string, error) {
19         if it.err != nil {
```

```
20                 return "", it.err
21         }
22         return it.it.Next(result)
23 }
24
25 type multiplexQuery struct {
26         err error
27         q   Query
28 }
29
30 func (q *multiplexQuery) Filter(field, comparator string, value
   ↪  interface{}) Query {
31         return &multiplexQuery{
32                 q:   q.q.Filter(field, comparator, value),
33                 err: q.err,
34         }
35 }
36
37 func (q *multiplexQuery) KeysOnly() Query {
38         return &multiplexQuery{
39                 q:   q.q.KeysOnly(),
40                 err: q.err,
41         }
42 }
43
44 func (q *multiplexQuery) Run() Iterator {
45         return &multiplexIterator{
46                 err: q.err,
47                 it:  q.q.Run(),
48         }
49 }
50
51 type multiplexBatch struct {
52         b Batch
53         s *multiplexStore
54 }
55
56 func (b *multiplexBatch) Set(table, key string, value interface{}) error
   ↪  {
```

```
57          return b.b.Set(table, key, value)
58  }
59
60  func (b *multiplexBatch) Commit() error {
61          err := b.b.Commit()
62          if err == nil {
63                  // If everything succeeds, we dump all the new state to
                    ↪  the underlying
64                  // store
65                  err = b.s.dump()
66          }
67          return err
68  }
69
70  type multiplexStore struct {
71          table string
72          key   string
73          cod   coder.Coder
74          s     Store
75          temp  *memoryStore
76  }
77
78  func (s *multiplexStore) dump() error {
79          buf := pool.GetBytesBuffer()
80          defer pool.PutBytesBuffer(buf)
81          err := s.cod.EncodeTo(&s.temp.data, buf)
82          if err == nil {
83                  val := MultiplexItem{
84                          Value: buf.Bytes(),
85                  }
86                  err = s.s.Set(s.table, s.key, &val)
87          }
88          return err
89  }
90
91  func (s *multiplexStore) load() error {
92          var val MultiplexItem
93          err := s.s.Get(s.table, s.key, &val)
94          var values map[string]map[string][]byte
```

```
95          if err == nil {
96                  err = s.cod.DecodeFrom(bytes.NewReader(val.Value),
                    ↪  &values)
97          }
98          if err == nil {
99                  s.temp.data = values
100         }
101         return err
102 }
103
104 // Get gets the specified record in the memory, putting it into result
105 func (s *multiplexStore) Get(table, key string, result interface{}) error
    ↪  {
106         err := s.load()
107         if err == nil {
108                 err = s.temp.Get(table, key, result)
109         }
110         return err
111 }
112
113 // GetMulti gets the specified records in the memory, putting it into
    ↪  result
114 func (s *multiplexStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
115         errLoad := s.load()
116         var err map[string]error
117         if errLoad == nil {
118                 err = s.temp.GetMulti(table, keys, results)
119         } else {
120                 err = make(map[string]error)
121                 for _, key := range keys {
122                         err[key] = errLoad
123                 }
124         }
125         return err
126 }
127
128 func (s *multiplexStore) GetAll(table string) Iterator {
129         return s.Query(table).Run()
```

```
130 }
131
132 func (s *multiplexStore) GetAllKeys(table string) Iterator {
133         return s.Query(table).KeysOnly().Run()
134 }
135
136 // Query returns a query manager to the specified table
137 func (s *multiplexStore) Query(table string) Query {
138         err := s.load()
139         q := s.temp.Query(table)
140         return &multiplexQuery{
141                 q:   q,
142                 err: err,
143         }
144 }
145
146 // Set puts the value on the table
147 func (s *multiplexStore) Set(table, key string, value interface{}) error
    ↪ {
148         err := s.temp.Set(table, key, value)
149         if err == nil {
150                 err = s.dump()
151         }
152         return err
153 }
154
155 // Del deletes the key from the table
156 func (s *multiplexStore) Del(table, key string) error {
157         err := s.temp.Del(table, key)
158         if err == nil {
159                 err = s.dump()
160         }
161         return err
162 }
163
164 // Batch returns an object that can batch updates to the store,
    ↪  automatically
165 func (s *multiplexStore) Batch() Batch {
166         return &multiplexBatch{
```

```
167                 b: s.temp.Batch(),
168                 s: s,
169         }
170 }
171
172 func (s *multiplexStore) Transaction(refs []Reference, function
    ↪  func(LimitedStore) error) error {
173         err := s.temp.Transaction(refs, function)
174         if err == nil {
175                 err = s.dump()
176         }
177         return err
178 }
179
180 func (s *multiplexStore) GenerateID(table string) (string, error) {
181         return s.temp.GenerateID(table)
182 }
183
184 // NewMultiplexStore returns a store that saves all its key in another
    ↪  store,
185 // in a specified table and key, coded with a coder.
186 // Because of the fact that for each operation the underlying store is
    ↪  touched
187 // You may want to use cachedStore as underlying store as it will avoid
    ↪  some
188 // repeated requests.
189 func NewMultiplexStore(s Store, cod coder.Coder, table, key string) Store
    ↪ {
190         cod.Register(MultiplexItem{})
191         memory := NewMemoryStore().(*memoryStore)
192         cod.Register(memory.data)
193         store := &multiplexStore{
194                 s:     s,
195                 table: table,
196                 key:   key,
197                 cod:   cod,
198                 temp:  memory,
199         }
200         return store
```

```
}
```

Arquivo overlay.go

```
package kv

import "sync"

type overlayBatch struct {
        canAccess func(table, key string) bool
        batch     Batch
        lock      sync.Mutex
        err       error
}

func (b *overlayBatch) Set(table, key string, value interface{}) error {
        b.lock.Lock()
        defer b.lock.Unlock()
        if b.err == nil && !b.canAccess(table, key) {
                b.err = errAccessDenied
        }
        if b.err == nil {
                b.err = b.batch.Set(table, key, value)
        }
        return b.err
}

func (b *overlayBatch) Commit() error {
        b.lock.Lock()
        defer b.lock.Unlock()
        if b.err == nil {
                b.err = b.batch.Commit()
        }
        return b.err
}

type overlayStore struct {
        refs  map[string]map[string]struct{}
        store LimitedStore
}
```

```
37
38 func (ov overlayStore) canAccess(table, key string) bool {
39         if tab, ok := ov.refs[table]; ok {
40                 _, ok = tab[key]
41                 return ok
42         }
43         return false
44 }
45 func (ov overlayStore) Get(table, key string, result interface{}) error {
46         if !ov.canAccess(table, key) {
47                 return errAccessDenied
48         }
49         return ov.store.Get(table, key, result)
50 }
51 func (ov overlayStore) GetMulti(table string, keys []string, results
   ↪  interface{}) map[string]error {
52         correctKeys := make([]string, 0, len(keys))
53         var deniedKeys []string
54         for _, key := range keys {
55                 if ov.canAccess(table, key) {
56                         correctKeys = append(correctKeys, key)
57                 } else {
58                         deniedKeys = append(deniedKeys, key)
59                 }
60         }
61         result := ov.store.GetMulti(table, correctKeys, results)
62         for _, key := range deniedKeys {
63                 result[key] = errAccessDenied
64         }
65         return result
66 }
67
68 func (ov overlayStore) Set(table, key string, value interface{}) error {
69         if !ov.canAccess(table, key) {
70                 return errAccessDenied
71         }
72         return ov.store.Set(table, key, value)
73 }
74
```

```
func (ov overlayStore) Del(table, key string) error {
        if !ov.canAccess(table, key) {
                return errAccessDenied
        }
        return ov.store.Del(table, key)
}

func (ov overlayStore) Batch() Batch {
        return &overlayBatch{
                batch:     ov.store.Batch(),
                canAccess: ov.canAccess,
        }
}

func (ov overlayStore) GenerateID(table string) (string, error) {
        return ov.store.GenerateID(table)
}

func refToMap(refs []Reference) map[string]map[string]struct{} {
        result := make(map[string]map[string]struct{})
        for _, ref := range refs {
                if _, ok := result[ref.Table]; !ok {
                        result[ref.Table] = make(map[string]struct{})
                }
                result[ref.Table][ref.Key] = struct{}{}
        }
        return result
}

func newOverlay(refs []Reference, store LimitedStore) LimitedStore {
        return overlayStore{
                refs:  refToMap(refs),
                store: store,
        }
}
```

Arquivo rollback.go

```
package kv

```

```
import (
        "fmt"
        "reflect"
        "sync"

        "github.com/pkg/errors"
)

type rollbackBatch struct {
        tmp   Batch
        types map[string]map[string]reflect.Type
        store *rollbackStore
        err   error
        lock  sync.Mutex
}

func (rs *rollbackBatch) Set(table, key string, value interface{}) error
↪ {
        rs.lock.Lock()
        defer rs.lock.Unlock()
        rv := reflect.ValueOf(value)
        if rs.err == nil && rv.Kind() != reflect.Ptr {
                rs.err = errValueMustBeAPointer
        }
        if rs.err == nil {
                rs.err = rs.tmp.Set(table, key, value)
        }
        if rs.err == nil {
                if _, ok := rs.types[table]; !ok {
                        rs.types[table] = make(map[string]reflect.Type)
                }
                rs.types[table][key] = rv.Elem().Type()
        }
        return rs.err
}

func (rs *rollbackBatch) Commit() error {
        rs.lock.Lock()
        defer rs.lock.Unlock()
```

```
41          if rs.err == nil {
42                  rs.err = rs.tmp.Commit()
43          }
44          // Save the new types of the keys in the store, as the commit
            ↪  succeeded
45          if rs.err == nil {
46                  for name, table := range rs.types {
47                          for key, typ := range table {
48                                  rs.store.learnType(name, key, typ)
49                          }
50                  }
51          }
52          return rs.err
53  }
54
55  type rollbackStore struct {
56          temporary   Store
57          store       LimitedStore
58          deletedLock sync.Mutex
59          typesLock   sync.Mutex
60          deleted     map[string]map[string]struct{}
61          types       map[string]map[string]reflect.Type
62  }
63
64  func (rs *rollbackStore) isDeleted(table, key string) bool {
65          rs.deletedLock.Lock()
66          defer rs.deletedLock.Unlock()
67          if tab, ok := rs.deleted[table]; ok {
68                  _, ok = tab[key]
69                  return ok
70          }
71          return false
72  }
73
74  func (rs *rollbackStore) learnType(table, key string, typ reflect.Type) {
75          rs.typesLock.Lock()
76          defer rs.typesLock.Unlock()
77          if _, ok := rs.types[table]; !ok {
78                  rs.types[table] = make(map[string]reflect.Type)
```

```
79          }
80          rs.types[table][key] = typ
81  }
82
83  func (rs *rollbackStore) learnTypeFromValue(table, key string, value
    ↪  interface{}) {
84          rs.learnType(table, key, reflect.ValueOf(value).Elem().Type())
85  }
86
87  func (rs *rollbackStore) Get(table, key string, result interface{}) error
    ↪  {
88          err := rs.store.Get(table, key, result)
89          if err == nil || IsKeyNotFoundError(err) {
90                  // KeyNotFound do not means that the type is incorrect..
91                  rs.learnTypeFromValue(table, key, result)
92          }
93          return err
94  }
95
96  func (rs *rollbackStore) GetMulti(table string, keys []string, results
    ↪  interface{}) map[string]error {
97          result := rs.store.GetMulti(table, keys, results)
98          for _, key := range keys {
99                  if result[key] == nil {
100                         rt := reflect.ValueOf(results).Type().Elem()
101                         if rt.Kind() == reflect.Ptr {
102                                 rt = rt.Elem()
103                         }
104                         rs.learnType(table, key, rt)
105                 }
106         }
107         return result
108 }
109
110 func (rs *rollbackStore) Set(table, key string, value interface{}) error
    ↪  {
111         if rs.isDeleted(table, key) {
112                 rs.deletedLock.Lock()
113                 delete(rs.deleted[table], key)
```

```
114                 rs.deletedLock.Unlock()
115         }
116         err := rs.temporary.Set(table, key, value)
117         if err == nil {
118                 rs.learnTypeFromValue(table, key, value)
119         }
120         return err
121 }
122
123 func (rs *rollbackStore) Del(table, key string) error {
124         rs.deletedLock.Lock()
125         if _, ok := rs.deleted[table]; !ok {
126                 rs.deleted[table] = make(map[string]struct{})
127         }
128         rs.deleted[table][key] = struct{}{}
129         rs.deletedLock.Unlock()
130         err := rs.temporary.Del(table, key)
131         return err
132 }
133
134 func (rs *rollbackStore) Batch() Batch {
135         return &rollbackBatch{
136                 tmp:   rs.temporary.Batch(),
137                 types: make(map[string]map[string]reflect.Type),
138                 store: rs,
139         }
140 }
141
142 func (rs *rollbackStore) GenerateID(table string) (string, error) {
143         return rs.store.GenerateID(table)
144 }
145
146 func (rs *rollbackStore) commit() error {
147         rs.typesLock.Lock()
148         rs.deletedLock.Lock()
149         defer rs.typesLock.Unlock()
150         defer rs.deletedLock.Unlock()
151         var err error
152         backup := NewMemoryStore()
```

```
153         var toRestore, toDelete []Reference
154         batch := rs.store.Batch()
155         for name, table := range rs.types {
156                 for key, typ := range table {
157                         rvalue := reflect.New(typ)
158                         value := rvalue.Interface()
159                         if err == nil {
160                                 // The 'value' created here is local and
                                    ↪  will not be reused
161                                 // by the method calls below
162                                 err = rs.temporary.Get(name, key, value)
163                         }
164                         if IsKeyNotFoundError(err) {
165                                 // Not exists on temporary, so there's no
                                    ↪  need to modify
166                                 // it...so...skip it!
167                                 err = nil
168                                 continue
169                         }
170                         rvalue.Elem().Set(reflect.Zero(typ))
171                         value = rvalue.Interface()
172                         if err == nil {
173                                 err = rs.store.Get(name, key, value)
174                         }
175                         if err == nil {
176                                 err = backup.Set(name, key, value)
177                         }
178                         ref := Reference{
179                                 Table: name,
180                                 Key:   key,
181                         }
182                         if IsKeyNotFoundError(err) {
183                                 toDelete = append(toDelete, ref)
184                                 err = nil // Ignores error because the
                                    ↪  register will be created
185                         } else {
186                                 toRestore = append(toRestore, ref)
187                         }
188                         rvalue.Elem().Set(reflect.Zero(typ))
```

```
189                         value = rvalue.Interface()
190                         if err == nil {
191                                 err = rs.temporary.Get(name, key, value)
192                         }
193                         if err == nil {
194                                 err = batch.Set(name, key, value)
195                         }
196                 }
197         }
198         if err == nil {
199                 err = batch.Commit()
200         }
201         for name, table := range rs.deleted {
202                 typesTable, tableExists := rs.types[name]
203                 if !tableExists && err == nil {
204                         err = fmt.Errorf("kv: not found type information
                        ↪  about table '%s'", name)
205                         continue
206                 }
207                 for key := range table {
208                         typ, exist := typesTable[key]
209                         if !exist {
210                                 if err == nil {
211                                         err = fmt.Errorf("kv: not found
                                        ↪  type information about table
                                        ↪  '%s' and key '%s'", name,
                                        ↪  key)
212                                 }
213                                 continue
214                         }
215                         rvalue := reflect.New(typ)
216                         value := rvalue.Interface()
217                         if err == nil {
218                                 err = rs.store.Get(name, key, value)
219                         }
220                         if err == nil {
221                                 err = backup.Set(name, key, value)
222                         }
223                         if err == nil {
```

```
224                             err = rs.store.Del(name, key)
225                     }
226                     if err == nil {
227                             toRestore = append(toRestore, Reference{
228                                     Table: name,
229                                     Key:   key,
230                             })
231                     }
232                     if IsKeyNotFoundError(err) {
233                             // Ignore the error (returned by
                                ↪  rs.store.Get) because
234                             // it means that the key already do not
                                ↪  exist
235                             err = nil
236                     }
237             }
238     }
239     if err != nil {
240             // Should rollback!
241             var errRollback error
242             batch = rs.store.Batch()
243             for _, ref := range toRestore {
244                     typ := rs.types[ref.Table][ref.Key]
245                     rvalue := reflect.New(typ)
246                     value := rvalue.Interface()
247                     if errRollback == nil {
248                             errRollback = rs.temporary.Get(ref.Table,
                                ↪  ref.Key, value)
249                     }
250                     if errRollback == nil {
251                             errRollback = batch.Set(ref.Table,
                                ↪  ref.Key, value)
252                     }
253             }
254             if errRollback == nil {
255                     errRollback = batch.Commit()
256             }
257             for _, ref := range toDelete {
258                     if errRollback == nil {
```

```
259                                 errRollback = rs.store.Del(ref.Table,
                                    ↪  ref.Key)
260                         }
261                 }
262                 if errRollback != nil {
263                         err = errors.Wrapf(err, "Error while doing
                            ↪  rollback: %s - Below is the original error:",
                            ↪  errRollback.Error())
264                 }
265         }
266         return err
267 }
268
269 func newRollbackStore(store LimitedStore) *rollbackStore {
270         return &rollbackStore{
271                 store:     store,
272                 temporary: NewMemoryStore(),
273                 deleted:   make(map[string]map[string]struct{}),
274                 types:     make(map[string]map[string]reflect.Type),
275         }
276 }
```

B.1.28 Pasta util/middlewares

Arquivo context.go

```
1  package middlewares
2
3  import (
4          "context"
5          "net/http"
6  )
7
8  type ctxType string
9
10 var requestType ctxType = "request-type"
11
12 func ContextRequest(next http.Handler) http.Handler {
13         return http.HandlerFunc(func(resp http.ResponseWriter, req
           ↪  *http.Request) {
```

```
14                  ctx := req.Context()
15                  ctx = context.WithValue(ctx, requestType, req)
16                  req = req.WithContext(ctx)
17                  next.ServeHTTP(resp, req)
18          })
19  }
20
21  func GetRequest(ctx context.Context) *http.Request {
22          return ctx.Value(requestType).(*http.Request)
23  }
```

B.1.29 Pasta util/notifications

Arquivo base.go

```
1   package notifications
2
3   import (
4           "encoding/gob"
5           "errors"
6           "time"
7   )
8
9   // ErrDone is returned when the iterator is totally exausted
10  var ErrDone = errors.New("notifications: iterator has no more results")
11
12  // Activity is a change on a entity of the system
13  type Activity struct {
14          Actor  string
15          Verb   string
16          Object string
17          Time   time.Time
18          Custom map[string]string
19  }
20
21  // Iterator is used to return the activities of an user feed or an entity
    ↪  feed
22  type Iterator interface {
23          Next() (Activity, error)
24  }
```

```
25
26 // StringIterator is used to return the IDs of the entities an user
   ↪  follows and
27 // the users that follows an entity
28 type StringIterator interface {
29         Next() (string, error)
30 }
31
32 // PublicNotifier impleemnts an interface that do not manage follows and
33 // followers and just send notifiers for everyone
34 type PublicNotifier interface {
35         Notify(activity Activity) error
36 }
37
38 // Notifier implements an interface that just manages follows and
   ↪  followers
39 // and sends notifications based on a activity and an entity
40 type Notifier interface {
41         PublicNotifier
42         AddFollower(entityID string, userID string) error
43         RemoveFollower(entityID string, userID string) error
44         GetFollowing(userID string, cursor string) StringIterator
45         GetFollowingCursors(userID string) StringIterator
46 }
47
48 // Feed implements an interface that implements an activities feed for
   ↪  the user
49 // and for the entities
50 type Feed interface {
51         Notifier
52         GetFeed(userID string) Iterator
53         GetEntityActivities(entityID string) Iterator
54 }
55
56 func init() {
57         gob.Register(notifierState{})
58         gob.Register(Activity{})
59         gob.Register([]Activity{})
60 }
```

Arquivo errors.go

```
1  package notifications
2
3  import (
4          "fmt"
5          "time"
6  )
7
8  type ErrDelay interface {
9          error
10         Delay() time.Duration
11 }
12 type errDelay struct {
13         parent error
14         delay  time.Duration
15 }
16
17 func (ed errDelay) Cause() error {
18         return ed.parent
19 }
20
21 func (ed errDelay) Error() string {
22         return fmt.Sprintf("Delaying '%s': %s", ed.delay.String(),
           ↪  ed.parent.Error())
23 }
24
25 func (ed errDelay) Delay() time.Duration {
26         return ed.delay
27 }
28
29 // NewErrDelay returns a error that implements ErrDelay
30 func NewErrDelay(parent error, delay time.Duration) ErrDelay {
31         return errDelay{parent, delay}
32 }
33
34 func getBestError(errorList []error) error {
35         var maxDelay time.Duration
36         var err error
```

```
37          for _, errOp := range errorList {
38                  if errOp == nil {
39                          // We tried every chat ID and at least ONE
                            ↪  returned successfully
40                          // So we can ignore the other errors here
41                          break
42                  }
43                  if delay, ok := errOp.(ErrDelay); ok && delay.Delay() >
                    ↪  maxDelay {
44                          maxDelay = delay.Delay()
45                  }
46                  err = errOp
47          }
48          if delay, ok := err.(ErrDelay); ok && delay.Delay() < maxDelay {
49                  type causeInterface interface {
50                          Cause() error
51                  }
52                  if errCause, okCause := err.(causeInterface); okCause &&
                    ↪  errCause.Cause() != nil {
53                          err = errCause.Cause()
54                  }
55          }
56          if maxDelay > 0 && err != nil {
57                  err = errDelay{
58                          parent: err,
59                          delay:  maxDelay,
60                  }
61          }
62          return err
63  }
```

Arquivo multiplex.go

```
1   package notifications
2
3   // Notify takes a slice of PublicNotifiers and call Notify on each one
4   func Notify(activity Activity, notifiers []PublicNotifier) error {
5           var err error
6           for _, notifier := range notifiers {
7                   if err == nil {
```

```
8                       err = notifier.Notify(activity)
9               }
10      }
11      return err
12  }
```

Arquivo notifier.go

```
1   package notifications
2
3   import (
4       "time"
5
6       "github.com/fjorgemota/gurudamatricula/util/pipeline"
7       "golang.org/x/net/context"
8   )
9
10  type notifierAdapter interface {
11      sendPublic(activity Activity) error
12      sendToUser(activity Activity, userID string) error
13  }
14
15  type notifierOptions struct {
16      DispatchWorkersTaskName       string
17      DispatchNotificationTaskName  string
18      DispatchNotificationsTaskName string
19      BatchSize                     int
20      MaxAttempts                   int
21  }
22
23  type notifierState struct {
24      Attempts int
25      UserIDs  []string
26      Activity Activity
27  }
28
29  type notifierDependencies struct {
30      followHandler PushFollowHandler
31      adapter       notifierAdapter
32  }
```

```
33
34 type notifierWorker struct {
35         getDependencies func(ctx context.Context) (notifierDependencies,
           ↪  error)
36         options         notifierOptions
37         getDelay        func() time.Duration
38 }
39
40 func (tw notifierWorker) sendNotification(pipe pipeline.Pipeline, attempt
   ↪  int, activity Activity, delay time.Duration, userIDs ...string) error
   ↪  {
41         _, err :=
           ↪  pipe.DispatchWithOptions(tw.options.DispatchNotificationTaskName,
           ↪  pipeline.TaskOptions{
42                 Delay: delay,
43         }, notifierState{
44                 Attempts: attempt,
45                 UserIDs:  userIDs,
46                 Activity: activity,
47         })
48         return err
49 }
50 func (tw notifierWorker) dispatchNotification(ctx context.Context, pipe
   ↪  pipeline.Pipeline, state notifierState) error {
51         dependencies, err := tw.getDependencies(ctx)
52         var errorsUserIDs []string
53         var maxDelay time.Duration
54         if err == nil {
55                 for _, userID := range state.UserIDs {
56                         errOp :=
                           ↪  dependencies.adapter.sendToUser(state.Activity,
                           ↪  userID)
57                         if errOp != nil {
58                                 errorsUserIDs = append(errorsUserIDs,
                                   ↪  userID)
59                         }
60                         if delay, ok := errOp.(ErrDelay); ok &&
                           ↪  delay.Delay() > maxDelay {
61                                 maxDelay = delay.Delay()
```

```
62                      }
63              }
64      }
65      if len(errorsUserIDs) > 0 && state.Attempts <
        ↪  tw.options.MaxAttempts {
66              err = tw.sendNotification(pipe, state.Attempts+1,
                ↪  state.Activity, maxDelay, errorsUserIDs...)
67      }
68      return err
69  }
70
71  func (tw notifierWorker) dispatchNotifications(ctx context.Context, pipe
    ↪  pipeline.Pipeline, activity Activity, cursor string) error {
72      dependencies, err := tw.getDependencies(ctx)
73      var followers StringIterator
74      if err == nil {
75              followers =
                ↪  dependencies.followHandler.GetFollowers(activity.Object,
                ↪  cursor)
76      }
77      var count int
78      limit := tw.options.BatchSize
79      if limit <= 0 {
80              limit = 1
81      }
82      batch := make([]string, 0, limit)
83      for err == nil && (limit <= 0 || count < limit) {
84              var userID string
85              userID, err = followers.Next()
86              if err == ErrDone {
87                      err = nil
88                      break
89              }
90              if len(batch) > limit && err == nil {
91                      err = tw.sendNotification(pipe, 0, activity, 0,
                        ↪  batch...)
92                      batch = batch[:0]
93              }
94              if err == nil {
```

```
95                          batch = append(batch, userID)
96                  }
97          }
98          if len(batch) > 0 && err == nil {
99                  err = tw.sendNotification(pipe, 0, activity, 0, batch...)
100         }
101         return err
102 }
103
104 func (tw notifierWorker) dispatchWorkers(ctx context.Context, pipe
    ↪  pipeline.Pipeline, activity Activity) error {
105         var err error
106         dependencies, err := tw.getDependencies(ctx)
107         if err == nil {
108                 err = dependencies.adapter.sendPublic(activity)
109         }
110         var cursorIt StringIterator
111         if err == nil {
112                 cursorIt =
                    ↪  dependencies.followHandler.GetFollowersCursors(activity.Object)
113         }
114         for err == nil {
115                 var cursor string
116                 cursor, err = cursorIt.Next()
117                 if err == ErrDone {
118                         err = nil
119                         break
120                 }
121                 if err == nil {
122                         _, err =
                            ↪  pipe.DispatchWithOptions(tw.options.DispatchNotificatio
                            ↪  pipeline.TaskOptions{
123                                 Delay: tw.getDelay(),
124                         }, activity, cursor)
125                 }
126         }
127         return err
128 }
129
```

```
130 // to dispatch the updates in a distributed manner
131 func registerNotifierWorker(dispatcher pipeline.Dispatcher, getDelay
    ↪  func() time.Duration, opts notifierOptions, getDependencies func(ctx
    ↪  context.Context) (notifierDependencies, error)) error {
132         if getDelay == nil {
133                 getDelay = func() time.Duration {
134                         return 0
135                 }
136         }
137         worker := notifierWorker{options: opts, getDependencies:
            ↪  getDependencies, getDelay: getDelay}
138         err := dispatcher.Register(opts.DispatchWorkersTaskName,
            ↪  worker.dispatchWorkers)
139         if err == nil {
140                 err =
                    ↪  dispatcher.Register(opts.DispatchNotificationsTaskName,
                    ↪  worker.dispatchNotifications)
141         }
142         if err == nil {
143                 err =
                    ↪  dispatcher.Register(opts.DispatchNotificationTaskName,
                    ↪  worker.dispatchNotification)
144         }
145         return err
146 }
```

Arquivo pull.go

```
1  package notifications
2
3  import (
4          "sort"
5  )
6
7  // PullHandler defines an interface to add and get activities from a
   ↪  entity
8  type PullHandler interface {
9          AddEntityActivity(activity Activity) error
10         GetEntityActivities(objectID string) Iterator
11 }
```

```
12
13  // PullFollowHandler defines an interface that manages follows
14  type PullFollowHandler interface {
15          AddFollower(entityID string, userID string) error
16          RemoveFollower(entityID string, userID string) error
17          GetFollowing(userID string, cursor string) StringIterator
18          GetFollowingCursors(userID string) StringIterator
19  }
20
21  type pullIterator struct {
22          userID              string
23          entitiesActivity    map[string]Activity
24          entitiesActivities map[string]Iterator
25          feed                Feed
26          booted              bool
27          entitiesOrder       []string
28  }
29
30  func (it *pullIterator) insertActivity(entity string, activity Activity)
    ↪ {
31          it.entitiesActivity[entity] = activity
32          i := sort.Search(len(it.entitiesOrder), func(n int) bool {
33                  return
                    ↪  it.entitiesActivity[it.entitiesOrder[n]].Time.Before(activity.T
                    ↪  ||
                    ↪  it.entitiesActivity[it.entitiesOrder[n]].Time.Equal(activity.Ti
34          })
35          it.entitiesOrder = append(it.entitiesOrder, "")
36          copy(it.entitiesOrder[i+1:], it.entitiesOrder[i:])
37          it.entitiesOrder[i] = entity
38  }
39
40  func (it *pullIterator) boot() error {
41          if it.booted {
42                  return nil
43          }
44          var err error
45          it.entitiesActivity = make(map[string]Activity)
46          it.entitiesActivities = make(map[string]Iterator)
```

```
47          cursorIt := it.feed.GetFollowingCursors(it.userID)
48          for err == nil {
49                  var cursor string
50                  cursor, err = cursorIt.Next()
51                  if err == ErrDone {
52                          err = nil
53                          break
54                  }
55                  entities := it.feed.GetFollowing(it.userID, cursor)
56                  for err == nil {
57                          var entity string
58                          entity, err = entities.Next()
59                          if err == ErrDone {
60                                  err = nil
61                                  break
62                          }
63                          iterator := it.feed.GetEntityActivities(entity)
64                          var activity Activity
65                          if err == nil {
66                                  activity, err = iterator.Next()
67                          }
68                          if err == nil {
69                                  it.entitiesActivities[entity] = iterator
70                                  it.insertActivity(entity, activity)
71                          }
72                          if err == ErrDone {
73                                  err = nil
74                          }
75                  }
76          }
77          it.booted = true
78          return err
79  }
80
81  func (it *pullIterator) Next() (Activity, error) {
82          err := it.boot()
83          var activity Activity
84          if len(it.entitiesOrder) == 0 {
85                  return activity, ErrDone
```

```
86          }
87          entity := it.entitiesOrder[0]
88          it.entitiesOrder = it.entitiesOrder[1:]
89          activity = it.entitiesActivity[entity]
90          var nextActivity Activity
91          if err == nil {
92                  nextActivity, err = it.entitiesActivities[entity].Next()
93          }
94          if err == nil {
95                  it.insertActivity(entity, nextActivity)
96          } else if err == ErrDone {
97                  // If the iterator from an activity returned ErrDone, we
                    ↪  can safely
98                  // ignore that error because there are other iterators
                    ↪  that we are
99                  // interested about, so we simply nil the value here and
                    ↪  let the
100                 // work continue
101                 err = nil
102         }
103         return activity, err
104
105 }
106
107 type pullFeed struct {
108         activitiesHandler PullHandler
109         followHandler     PullFollowHandler
110 }
111
112 func (pf pullFeed) Notify(activity Activity) error {
113         return pf.activitiesHandler.AddEntityActivity(activity)
114 }
115
116 func (pf pullFeed) GetEntityActivities(entityID string) Iterator {
117         return pf.activitiesHandler.GetEntityActivities(entityID)
118 }
119
120 func (pf pullFeed) GetFeed(userID string) Iterator {
121         return &pullIterator{
```

```
122                 userID: userID,
123                 feed:   pf,
124         }
125 }
126
127 func (pf pullFeed) AddFollower(entityID string, userID string) error {
128         return pf.followHandler.AddFollower(entityID, userID)
129 }
130
131 func (pf pullFeed) RemoveFollower(entityID string, userID string) error {
132         return pf.followHandler.RemoveFollower(entityID, userID)
133 }
134
135 func (pf pullFeed) GetFollowing(userID string, cursor string)
    ↪  StringIterator {
136         return pf.followHandler.GetFollowing(userID, cursor)
137 }
138
139 func (pf pullFeed) GetFollowingCursors(userID string) StringIterator {
140         return pf.followHandler.GetFollowingCursors(userID)
141 }
142
143 // NewPullFeed returns a Feed that pulls activities to the user, so
    ↪  Notify is
144 // CPU light and GetFeed is more...CPU intensive.
145 func NewPullFeed(activitiesHandler PullHandler, followHandler
    ↪  PullFollowHandler) Feed {
146         return pullFeed{activitiesHandler: activitiesHandler,
          ↪  followHandler: followHandler}
147 }
```

Arquivo push.go

```
1 package notifications
2
3 import (
4         "github.com/fjorgemota/gurudamatricula/util/pipeline"
5         "golang.org/x/net/context"
6 )
7
```

```
8  type PushHandler interface {
9          PullHandler
10         AddUserActivity(activity Activity, userID string) error
11         AddUserActivities(user, entityID string) error
12         RemoveUserActivities(userID, entityID string) error
13         GetUserActivities(userID string) Iterator
14 }
15
16 type PushFollowHandler interface {
17         PullFollowHandler
18         GetFollowers(entityID string, cursor string) StringIterator
19         GetFollowersCursors(entityID string) StringIterator
20 }
21
22 type pushFeed struct {
23         pipe            pipeline.Pipeline
24         activityHandler PushHandler
25         followHandler   PushFollowHandler
26         options         PushFeedOptions
27 }
28
29 func (pf pushFeed) Notify(activity Activity) error {
30         _, err := pf.pipe.Dispatch(pf.options.DispatchWorkersTaskName,
           ↪  activity)
31         return err
32 }
33
34 func (pf pushFeed) GetFeed(userID string) Iterator {
35         return pf.activityHandler.GetUserActivities(userID)
36 }
37
38 func (pf pushFeed) GetEntityActivities(entityID string) Iterator {
39         return pf.activityHandler.GetEntityActivities(entityID)
40 }
41
42 func (pf pushFeed) AddFollower(entityID, userID string) error {
43         err := pf.followHandler.AddFollower(entityID, userID)
44         if err == nil {
```

```
45                  err = pf.activityHandler.AddUserActivities(userID,
                    ↪  entityID)
46          }
47          return err
48  }
49
50  func (pf pushFeed) RemoveFollower(entityID, userID string) error {
51          err := pf.followHandler.RemoveFollower(entityID, userID)
52          if err == nil {
53                  err = pf.activityHandler.RemoveUserActivities(userID,
                    ↪  entityID)
54          }
55          return err
56  }
57
58  func (pf pushFeed) GetFollowing(userID, cursor string) StringIterator {
59          return pf.followHandler.GetFollowing(userID, cursor)
60  }
61
62  func (pf pushFeed) GetFollowingCursors(userID string) StringIterator {
63          return pf.followHandler.GetFollowingCursors(userID)
64  }
65
66  // PushFeedDependencies is a struct representing all the things a
    ↪  PushFeed
67  // needs to work, like access to the follow database and the hability to
68  // publish things on activity database
69  type PushFeedDependencies struct {
70          Activity PushHandler
71          Follow   PushFollowHandler
72  }
73
74  // NewPushFeed returns a feed that pushes activities to users instead of
    ↪  pull
75  // activities for user. That means that the biggest activity run on each
    ↪  Notify
76  // call, and not on GetFeed call. So the user can see the feed many times
77  // without a large use of the CPU power.
```

```
78  func NewPushFeed(pipe pipeline.Pipeline, dep PushFeedDependencies,
    ↪  options PushFeedOptions) Feed {
79          options = getDefaultPushFeedOptions(options)
80          return pushFeed{activityHandler: dep.Activity, options: options,
            ↪  followHandler: dep.Follow, pipe: pipe}
81  }
82
83  type PushFeedOptions struct {
84          DispatchWorkersTaskName       string
85          DispatchNotificationsTaskName string
86          MaxAttempts                   int
87  }
88
89  func getDefaultPushFeedOptions(opts PushFeedOptions) PushFeedOptions {
90          if opts.DispatchWorkersTaskName == "" {
91                  opts.DispatchWorkersTaskName =
                    ↪  "push_feed.dispatch_workers"
92          }
93          if opts.DispatchNotificationsTaskName == "" {
94                  opts.DispatchNotificationsTaskName =
                    ↪  "push_feed.dispatch_notifications"
95          }
96          if opts.MaxAttempts == 0 {
97                  opts.MaxAttempts = 3
98          }
99          return opts
100 }
101
102 type pushFeedAdapter struct {
103         activityHandler PushHandler
104 }
105
106 func (pfa pushFeedAdapter) sendToUser(activity Activity, userID string)
    ↪  error {
107         return pfa.activityHandler.AddUserActivity(activity, userID)
108 }
109 func (pfa pushFeedAdapter) sendPublic(activity Activity) error {
110         return pfa.activityHandler.AddEntityActivity(activity)
111 }
```

```
112
113 // RegisterPushFeedWorker registers functions on Pipeline Dispatcher that
    ↪  allows
114 // to dispatch the updates in a distributed manner
115 func RegisterPushFeedWorker(dispatcher pipeline.Dispatcher, opts
    ↪  PushFeedOptions, getDependencies func(ctx context.Context)
    ↪  (PushFeedDependencies, error)) error {
116         opts = getDefaultPushFeedOptions(opts)
117         return registerNotifierWorker(dispatcher, nil, notifierOptions{
118                 DispatchNotificationsTaskName:
                    ↪  opts.DispatchNotificationsTaskName,
119                 DispatchWorkersTaskName:
                    ↪  opts.DispatchWorkersTaskName,
120                 MaxAttempts:                    opts.MaxAttempts,
121         }, func(ctx context.Context) (notifierDependencies, error) {
122                 deps, err := getDependencies(ctx)
123                 return notifierDependencies{
124                         adapter:       pushFeedAdapter{deps.Activity},
125                         followHandler: deps.Follow,
126                 }, err
127         })
128 }
```

Arquivo telegram.go

```
1  package notifications
2
3  import (
4          "math/rand"
5          "time"
6
7          "github.com/fjorgemota/gurudamatricula/util/clock"
8          "github.com/fjorgemota/gurudamatricula/util/pipeline"
9          "golang.org/x/net/context"
10         "gopkg.in/telegram-bot-api.v4"
11 )
12
13 type TelegramFollowHandler interface {
14         PushFollowHandler
15         GetTelegramChatIDs(userID string) ([]int64, error)
```

```
}
type telegramNotifier struct {
        followHandler TelegramFollowHandler
        options       TelegramOptions
        pipe          pipeline.Pipeline
}

func (n telegramNotifier) Notify(activity Activity) error {
        _, err := n.pipe.Dispatch(n.options.DispatchWorkersTaskName,
        ↪  activity)
        return err
}

func (n telegramNotifier) AddFollower(entityID, userID string) error {
        return n.followHandler.AddFollower(entityID, userID)
}

func (n telegramNotifier) RemoveFollower(entityID, userID string) error {
        return n.followHandler.RemoveFollower(entityID, userID)
}

func (n telegramNotifier) GetFollowing(userID, cursor string)
↪  StringIterator {
        return n.followHandler.GetFollowing(userID, cursor)
}

func (n telegramNotifier) GetFollowingCursors(userID string)
↪  StringIterator {
        return n.followHandler.GetFollowingCursors(userID)
}

type TelegramDependencies struct {
        FollowHandler TelegramFollowHandler
        Bot           TelegramBot
        Format        func(activity Activity, userID string) (string,
        ↪  error)
}

type TelegramOptions struct {
```

```
51          DispatchWorkersTaskName       string
52          DispatchNotificationsTaskName string
53          Channel                       string
54          MaxAttempts                   int
55  }
56
57  func getDefaultTelegramOptions(opts TelegramOptions) TelegramOptions {
58          if opts.DispatchWorkersTaskName == "" {
59                  opts.DispatchWorkersTaskName =
                    ↪  "telegram.dispatch_workers"
60          }
61          if opts.DispatchNotificationsTaskName == "" {
62                  opts.DispatchNotificationsTaskName =
                    ↪  "telegram.dispatch_notifications"
63          }
64          if opts.MaxAttempts == 0 {
65                  opts.MaxAttempts = 3
66          }
67          return opts
68  }
69
70  type TelegramBot interface {
71          Send(c tgbotapi.Chattable) (tgbotapi.Message, error)
72  }
73
74  type telegramAdapter struct {
75          dependencies TelegramDependencies
76          options      TelegramOptions
77  }
78
79  func (ta telegramAdapter) sendPublic(activity Activity) error {
80          message, err := ta.dependencies.Format(activity, "")
81          if ta.options.Channel != "" {
82                  msg := tgbotapi.NewMessageToChannel(ta.options.Channel,
                    ↪  message)
83                  _, err = ta.dependencies.Bot.Send(msg)
84          }
85          return err
86  }
```

```
87
88  func (ta telegramAdapter) sendToUser(activity Activity, userID string)
    ↪  error {
89          message, err := ta.dependencies.Format(activity, userID)
90          var chats []int64
91          if err == nil {
92                  chats, err =
                    ↪  ta.dependencies.FollowHandler.GetTelegramChatIDs(userID)
93          }
94          if err == nil {
95                  var errors []error
96                  for _, chat := range chats {
97                          msg := tgbotapi.NewMessage(chat, message)
98                          _, errOp := ta.dependencies.Bot.Send(msg)
99                          if errResponse, ok := errOp.(tgbotapi.Error); ok
                            ↪  {
100                                 errOp = NewErrDelay(errResponse,
                                    ↪  time.Second*time.Duration(errResponse.RetryAfte
101                         }
102                         errors = append(errors, errOp)
103                 }
104                 err = getBestError(errors)
105         }
106         return err
107 }
108
109 // NewTelegramNotifier creates a notifier that sends notifications via
    ↪  telegram
110 func NewTelegramNotifier(pipe pipeline.Pipeline, followHandler
    ↪  TelegramFollowHandler, options TelegramOptions) Notifier {
111         options = getDefaultTelegramOptions(options)
112         return telegramNotifier{
113                 options:       options,
114                 pipe:          pipe,
115                 followHandler: followHandler,
116         }
117 }
118
```

```
119 // RegisterTelegramWorker registers functions on Pipeline Dispatcher that
    ↪ allows
120 // to dispatch the updates in a distributed manner
121 func RegisterTelegramWorker(dispatcher pipeline.Dispatcher, clk
    ↪ clock.Clock, opts TelegramOptions, getDependencies func(ctx
    ↪ context.Context) (TelegramDependencies, error)) error {
122         opts = getDefaultTelegramOptions(opts)
123         source := rand.NewSource(clk.Now().Unix())
124         gen := rand.New(source)
125         return registerNotifierWorker(dispatcher, func() time.Duration {
126                 return time.Second * time.Duration(gen.Int63n(12*3600))
127         }, notifierOptions{
128                 DispatchNotificationsTaskName:
                    ↪ opts.DispatchNotificationsTaskName,
129                 DispatchWorkersTaskName:
                    ↪ opts.DispatchWorkersTaskName,
130                 MaxAttempts:                    opts.MaxAttempts,
131         }, func(ctx context.Context) (notifierDependencies, error) {
132                 deps, err := getDependencies(ctx)
133                 return notifierDependencies{
134                         adapter:       telegramAdapter{dependencies: deps,
                            ↪ options: opts},
135                         followHandler: deps.FollowHandler,
136                 }, err
137         })
138 }
```

Arquivo webpush.go

```
1  package notifications
2
3  import (
4          "errors"
5          "net/http"
6          "time"
7
8          webpush "github.com/SherClockHolmes/webpush-go"
9          "github.com/fjorgemota/gurudamatricula/util/clock"
10         "github.com/fjorgemota/gurudamatricula/util/pipeline"
11         "golang.org/x/net/context"
```

```
12 )
13
14 // WebPushFollowHandler is a PushFollowHandler that is able to Get
   ↪ WebPush
15 // subscriptions for a UserID too
16 type WebPushFollowHandler interface {
17         PushFollowHandler
18         GetWebPushSubscriptions(userID string) ([]*webpush.Subscription,
           ↪ error)
19         RemoveWebPushSubscription(userID string, subscription
           ↪ *webpush.Subscription) error
20 }
21
22 type WebPushOptions struct {
23         DispatchWorkersTaskName       string
24         DispatchNotificationsTaskName string
25         VAPIDPrivateKey               string
26         VAPIDSubscriber               string
27         MaxAttempts                   int
28 }
29
30 func getDefaultWebPushOptions(opts WebPushOptions) WebPushOptions {
31         if opts.DispatchWorkersTaskName == "" {
32                 opts.DispatchWorkersTaskName = "webpush.dispatch_workers"
33         }
34         if opts.DispatchNotificationsTaskName == "" {
35                 opts.DispatchNotificationsTaskName =
                   ↪ "webpush.dispatch_notifications"
36         }
37         if opts.MaxAttempts == 0 {
38                 opts.MaxAttempts = 3
39         }
40         return opts
41 }
42
43 type webPushNotifier struct {
44         followHandler WebPushFollowHandler
45         options       WebPushOptions
46         pipe          pipeline.Pipeline
```

```
47 }
48
49 func (n webPushNotifier) Notify(activity Activity) error {
50         _, err := n.pipe.Dispatch(n.options.DispatchWorkersTaskName,
           ↪  activity)
51         return err
52 }
53
54 func (n webPushNotifier) AddFollower(entityID, userID string) error {
55         return n.followHandler.AddFollower(entityID, userID)
56 }
57 func (n webPushNotifier) RemoveFollower(entityID, userID string) error {
58         return n.followHandler.RemoveFollower(entityID, userID)
59 }
60 func (n webPushNotifier) GetFollowing(userID string, cursor string)
   ↪  StringIterator {
61         return n.followHandler.GetFollowing(userID, cursor)
62 }
63 func (n webPushNotifier) GetFollowingCursors(userID string)
   ↪  StringIterator {
64         return n.followHandler.GetFollowingCursors(userID)
65 }
66
67 // NewWebPushNotifier is a notifier that implements the Push API protocol
   ↪  and,
68 // because of that, it is able to notify browsers in realtime
69 func NewWebPushNotifier(pipe pipeline.Pipeline, followHandler
   ↪  WebPushFollowHandler, options WebPushOptions) Notifier {
70         options = getDefaultWebPushOptions(options)
71         return webPushNotifier{
72                 followHandler: followHandler,
73                 pipe:          pipe,
74                 options:       options,
75         }
76 }
77
78 type WebPushDependencies struct {
79         Client           *http.Client
80         FollowHandler    WebPushFollowHandler
```

```
81          Format           func(activity Activity, userID string) (string,
            ↪  error)
82          SendNotification func(message []byte, s *webpush.Subscription,
            ↪  options *webpush.Options) (*http.Response, error)
83  }
84  type webpushAdapter struct {
85          dependencies WebPushDependencies
86          options      WebPushOptions
87  }
88
89  func (wpa webpushAdapter) parseDuration(retryAfter string) (time.Duration,
    ↪  error) {
90          duration, err := time.ParseDuration(retryAfter + "s")
91          if err == nil {
92                  return duration, err
93          }
94          parsed, err := http.ParseTime(retryAfter)
95          if err == nil {
96                  duration = parsed.Sub(time.Now())
97          }
98          return duration, err
99
100 }
101 func (wpa webpushAdapter) sendPublic(activity Activity) error {
102         return nil
103 }
104
105 func (wpa webpushAdapter) sendToUser(activity Activity, userID string)
    ↪  error {
106         var subscriptions []*webpush.Subscription
107         message, err := wpa.dependencies.Format(activity, userID)
108         if err == nil {
109                 subscriptions, err =
                    ↪  wpa.dependencies.FollowHandler.GetWebPushSubscriptions(userID)
110         }
111         if err == nil {
112                 var errorsList []error
113                 for _, subscription := range subscriptions {
```

```
114                         response, errOp :=
                            ↪  wpa.dependencies.SendNotification([]byte(message),
                            ↪  subscription, &webpush.Options{
115                                 HTTPClient:      wpa.dependencies.Client,
116                                 VAPIDPrivateKey:
                                    ↪  wpa.options.VAPIDPrivateKey,
117                                 Subscriber:
                                    ↪  wpa.options.VAPIDSubscriber,
118                         })
119                         if errOp == nil && response != nil &&
                            ↪  response.StatusCode != 201 {
120                                 errOp = errors.New("notifications:
                                    ↪  Invalid HTTP Status")
121                         }
122
123                         if response != nil && errOp != nil &&
                            ↪  (response.StatusCode == 410 ||
                            ↪  response.StatusCode == 404) {
124                                 errOp =
                                    ↪  wpa.dependencies.FollowHandler.RemoveWebPushSubscrip
                                    ↪  subscription)
125                         }
126                         if response != nil && errOp != nil &&
                            ↪  response.StatusCode == 429 {
127                                 var duration time.Duration
128                                 duration, errOp =
                                    ↪  wpa.parseDuration(response.Header.Get("Retry-After")
129                                 if errOp == nil {
130                                         errOp = errors.New("the request
                                            ↪  to webpush should be
                                            ↪  retried")
131                                         errOp = NewErrDelay(errOp,
                                            ↪  duration)
132                                 }
133                         }
134                         errorsList = append(errorsList, errOp)
135                 }
136                 err = getBestError(errorsList)
137         }
```

```
138         return err
139 }
140
141 // RegisterWebpushWorker registers functions on Pipeline Dispatcher that
    ↪  allows
142 // to dispatch the updates in a distributed manner
143 func RegisterWebpushWorker(dispatcher pipeline.Dispatcher, clk
    ↪  clock.Clock, opts WebPushOptions, getDependencies func(ctx
    ↪  context.Context) (WebPushDependencies, error)) error {
144         opts = getDefaultWebPushOptions(opts)
145         return registerNotifierWorker(dispatcher, nil, notifierOptions{
146                 DispatchNotificationsTaskName:
                    ↪  opts.DispatchNotificationsTaskName,
147                 DispatchWorkersTaskName:
                    ↪  opts.DispatchWorkersTaskName,
148                 MaxAttempts:                    opts.MaxAttempts,
149         }, func(ctx context.Context) (notifierDependencies, error) {
150                 deps, err := getDependencies(ctx)
151                 if err == nil {
152                         if deps.Client == nil {
153                                 deps.Client = http.DefaultClient
154                         }
155                         if deps.SendNotification == nil {
156                                 deps.SendNotification =
                                    ↪  webpush.SendNotification
157                         }
158                 }
159                 return notifierDependencies{
160                         adapter:       webpushAdapter{dependencies: deps,
                            ↪  options: opts},
161                         followHandler: deps.FollowHandler,
162                 }, err
163         })
164 }
```

B.1.30 Pasta util/pipeline

Arquivo base.go

```
1  package pipeline
2
3  import (
4          "errors"
5          "io"
6          "reflect"
7          "time"
8
9          "golang.org/x/net/context"
10 )
11
12 var (
13         ErrFutureNotFound     = errors.New("pipeline: Future not found")
14         ErrInvalidStatus      = errors.New("pipeline: Invalid status")
15         ErrInvalidOutputIndex = errors.New("pipeline: Invalid output
           ↪  index")
16 )
17
18 // Pipeline allows to register functions and call these functions as
   ↪  futures
19 type Pipeline interface {
20         Dispatch(task string, args ...interface{}) ([]FutureID, error)
21         DispatchWithOptions(task string, options TaskOptions, args
           ↪  ...interface{}) ([]FutureID, error)
22         GetResults(futID string) ([]interface{}, error)
23         GetResult(futID FutureID) (interface{}, error)
24         GetStatus(futID string) (FutureStatus, error)
25         DeleteFinished(futID string) error
26 }
27
28 // Typer allow to gets the output type of a FutureID
29 type Typer interface {
30         GetTaskOutputType(futID FutureID) (reflect.Type, error)
31 }
32
33 // Worker implements an interface
34 type Worker interface {
35         Work(ctx context.Context, futID string) error
36         ParentDone(ctx context.Context, parentID, childID string) error
```

```
37 }
38
39 // Dispatcher encodes and decodes calls to tasks
40 type Dispatcher interface {
41         Register(name string, fn interface{}) error
42         Decode(ctx context.Context, caller, pipeline Pipeline, future
   ↪  *Future, reader io.Reader) (bool, error)
43         Encode(writer io.Writer, name string, typer Typer, args
   ↪  ...interface{}) error
44         GetType(name string) (reflect.Type, error)
45 }
46
47 //go:generate stringer -type=FutureStatus -output=future_status_string.go
48
49 // FutureStatus represents the status of the future
50 type FutureStatus int64
51
52 const (
53         WaitingDependency FutureStatus = 1 << (1 + iota)
54         WaitingQueue
55         WaitingChildren
56         Running
57         Done
58         Aborted
59 )
60
61 // DefaultMaxAttempts is the number of maximum attempts that the system
   ↪  will
62 // try to execute
63 const DefaultMaxAttempts = 3
64
65 // FutureID represents an output for a future
66 type FutureID struct {
67         ID     string
68         Output int
69 }
70
71 // DependencyArg is a representation between a dependency with an
   ↪  argument in a
```

```
72 // Future
73 type DependencyArg struct {
74         Argument   int `datastore:"argument,noindex" json:"argument"`
75         Dependency int `datastore:"dependency,noindex" json:"dependency"`
76 }
77
78 // Future represents a task that will be completed in the future, with
   ↪  all it's
79 // dependencies and dependents correctly registered
80 type Future struct {
81         Parent                          string
          ↪  `datastore:"parent,noindex" json:"parent"`
82         Name                            string
          ↪  `datastore:"name,noindex" json:"name"`
83         Data                            []byte
          ↪  `datastore:"data,noindex" json:"data"`
84         Children                        []string
          ↪  `datastore:"children,noindex" json:"children"`
85         ChildrenWithoutDependents       []string
          ↪  `datastore:"children_without_dependents,noindex"
          ↪  json:"children_without_dependents"`
86         ChildrenWithoutDependentsDone []string
          ↪  `datastore:"children_without_dependents_done,noindex"
          ↪  json:"children_without_dependents_done"`
87         Dependencies                    []FutureID
          ↪  `datastore:"dependencies,noindex" json:"dependencies"`
88         DependencyArgs                  []DependencyArg
          ↪  `datastore:"dependency_args,noindex" json:"dependency_args"`
89         DependenciesDone                []string
          ↪  `datastore:"dependencies_done,noindex"
          ↪  json:"dependencies_done"`
90         Dependents                      []string
          ↪  `datastore:"dependents,noindex" json:"dependents"`
91         Status                          FutureStatus
          ↪  `datastore:"status,noindex" json:"status"`
92         Results                         []byte
          ↪  `datastore:"results,noindex" json:"results"`
93         Attempts                        int
          ↪  `datastore:"attempts,noindex" json:"attempts"`
```

```
94          MaxAttempts                     int
            ↪  `datastore:"max_attempts,noindex" json:"max_attempts"`
95          Delay                           time.Duration
            ↪  `datastore:"delay,noindex" json:"delay"`
96          RegisteredAt                    time.Time
            ↪  `datastore:"registered_at,noindex" json:"registered_at"`
97          StartedAt                       time.Time
            ↪  `datastore:"started_at,noindex" json:"started_at"`
98          FinishedAt                      time.Time
            ↪  `datastore:"finished_at,noindex" json:"finished_at"`
99          AbortedAt                       time.Time
            ↪  `datastore:"aborted_at,noindex" json:"aborted_at"`
100 }
101
102 // GlobalOptions represents a few string options that the mechanism uses,
    ↪  like
103 // table and endpoints for Work and Parent Done tasks
104 type GlobalOptions struct {
105         Table              string
106         WorkTaskName       string
107         ParentDoneTaskName string
108         AbortTaskName      string
109         MaxAttempts        int
110 }
111
112 // TaskOptions are options that are applied by task, so it's possible to
    ↪  control
113 // things like MaxAttempts of the task and delay to execute the task
114 type TaskOptions struct {
115         MaxAttempts int
116         Delay       time.Duration
117         Detach      bool
118 }
```

Arquivo default.go

```
1  package pipeline
2
3  func getDefaultOptions(options GlobalOptions) GlobalOptions {
4          if options.Table == "" {
```

```
5                 options.Table = "pipeline_tasks"
6         }
7         if options.ParentDoneTaskName == "" {
8                 options.ParentDoneTaskName = "pipeline_done"
9         }
10        if options.WorkTaskName == "" {
11                options.WorkTaskName = "pipeline_work"
12        }
13        if options.AbortTaskName == "" {
14                options.AbortTaskName = "pipeline_abort"
15        }
16        if options.MaxAttempts == 0 {
17                options.MaxAttempts = DefaultMaxAttempts
18        }
19        if options.MaxAttempts < 0 {
20                options.MaxAttempts = 0
21        }
22        return options
23 }
```

Arquivo dispatcher.go

```
1  package pipeline
2
3  import (
4         "io"
5         "reflect"
6
7         "golang.org/x/net/context"
8
9         "github.com/fjorgemota/gurudamatricula/util/coder"
10        "github.com/fjorgemota/gurudamatricula/util/dispatcher"
11 )
12
13 type pipelineDispatcher struct {
14        d dispatcher.Dispatcher
15 }
16
17 func (fd pipelineDispatcher) Register(name string, fn interface{}) error
   ↪ {
```

```
18          return fd.d.Register(name, fn)
19  }
20
21  func (fd pipelineDispatcher) Encode(writer io.Writer, name string, typer
    ↪  Typer, args ...interface{}) error {
22          newArgs := make([]interface{}, 0, len(args)+1)
23          newArgs = append(newArgs, typer)
24          newArgs = append(newArgs, args...)
25          return fd.d.Encode(writer, name, newArgs...)
26  }
27
28  func (fd pipelineDispatcher) Decode(ctx context.Context, caller, pipeline
    ↪  Pipeline, future *Future, reader io.Reader) (bool, error) {
29          return fd.d.Decode(reader, ctx, caller, pipeline, future)
30  }
31
32  func (fd pipelineDispatcher) GetType(name string) (reflect.Type, error) {
33          return fd.d.GetType(name)
34  }
35
36  // NewDispatcher returns a new pipeline dispatcher
37  func NewDispatcher(cod coder.Coder) Dispatcher {
38          return pipelineDispatcher{
39                  d: dispatcher.NewDispatcher(cod, handler{}),
40          }
41  }
```

Arquivo future_status_string.go

```
1   // Code generated by "stringer -type=FutureStatus
    ↪  -output=future_status_string.go"; DO NOT EDIT.
2
3   package pipeline
4
5   import "strconv"
6
7   const (
8           _FutureStatus_name_0 = "WaitingDependency"
9           _FutureStatus_name_1 = "WaitingQueue"
10          _FutureStatus_name_2 = "WaitingChildren"
```

```
11          _FutureStatus_name_3 = "Running"
12          _FutureStatus_name_4 = "Done"
13          _FutureStatus_name_5 = "Aborted"
14  )
15
16  func (i FutureStatus) String() string {
17          switch {
18          case i == 2:
19                  return _FutureStatus_name_0
20          case i == 4:
21                  return _FutureStatus_name_1
22          case i == 8:
23                  return _FutureStatus_name_2
24          case i == 16:
25                  return _FutureStatus_name_3
26          case i == 32:
27                  return _FutureStatus_name_4
28          case i == 64:
29                  return _FutureStatus_name_5
30          default:
31                  return "FutureStatus(" + strconv.FormatInt(int64(i), 10)
                    ↪  + ")"
32          }
33  }
```

Arquivo handler.go

```
1   package pipeline
2
3   import (
4           "errors"
5           "fmt"
6           "reflect"
7
8           "github.com/fjorgemota/gurudamatricula/util/dispatcher"
9           "golang.org/x/net/context"
10  )
11
12  var contextType = reflect.TypeOf((*context.Context)(nil)).Elem()
13  var pipelineType = reflect.TypeOf((*Pipeline)(nil)).Elem()
```

```
14 var futureIDType = reflect.TypeOf(FutureID{})
15 var errorType = reflect.TypeOf((*error)(nil)).Elem()
16
17 type handler struct{}
18
19 func (h handler) ValidateFunction(ft reflect.Type) error {
20         if ft.NumIn() == 0 || ft.In(0) != contextType {
21                 return errors.New("first argument of fn should be a
                   ↪  context")
22         }
23         if ft.NumIn() == 1 || ft.In(1) != pipelineType {
24                 return errors.New("second argument of fn should be a
                   ↪  Pipeline")
25         }
26         if ft.NumOut() == 0 || ft.Out(ft.NumOut()-1) != errorType {
27                 return errors.New("the function should have at least an
                   ↪  error output")
28         }
29         return nil
30 }
31 func (h handler) ProcessCall(ft reflect.Type, args []interface{})
   ↪  ([]interface{}, error) {
32         var err error
33         types := make([]interface{}, len(args))
34         copy(types, args)
35         r := args[0].(Typer)
36         // Cut register from the list of arguments
37         args = args[1:]
38         types = types[1:]
39         // Map FutureIDs for it's correct types
40         for i, arg := range args {
41                 if futID, ok := arg.(FutureID); ok {
42                         var futType reflect.Type
43                         futType, err = r.GetTaskOutputType(futID)
44                         if err == nil {
45                                 if reflect.TypeOf(futID) == futType {
46                                         // The return type is a Future..
47                                         // So guess that the final type
                                           ↪  is correct and
```

```
                                        // ignore the output type for a moment
                                        arg := i + 2
                                        variadic := false
                                        if ft.IsVariadic() && arg >= ft.NumIn()-1 {
                                                arg = ft.NumIn() - 1
                                                variadic = true
                                        }
                                        futType = ft.In(arg)
                                        if variadic {
                                                futType = futType.Elem()
                                        }
                                }
                                // It's a future! Get a zero value of the type
                                types[i] = reflect.Zero(futType).Interface()
                        }
                }
        }
        // Assume context.Context and Pipeline as arguments too
        nArgs := len(args) + 2
        if err == nil {
                err = dispatcher.ValidateMinArguments(ft, nArgs)
        }
        if err == nil {
                err = dispatcher.ValidateMaxArguments(ft, nArgs)
        }
        var fnArgs []reflect.Type
        if err == nil {
                fnArgs, err = dispatcher.GetArgumentTypes(ft)
        }
        if err == nil {
                fnArgs = fnArgs[2:]
        }
        if err == nil && ft.IsVariadic() && nArgs >= len(fnArgs)-1 {
                fnArgs = dispatcher.AdaptVariadic(fnArgs, len(args))
        }
```

```
83          if err == nil {
84                  err = dispatcher.ValidateCompatibleTypes(fnArgs, types)
85          }
86          return args, err
87  }
88  func (h handler) ProcessArgs(ft reflect.Type, oldArgs, newArgs
    ↪  []interface{}) ([]interface{}, error) {
89          args := make([]interface{}, 0, len(oldArgs)+2)
90          args = append(args, newArgs[0])
91          args = append(args, newArgs[1])
92          fut := newArgs[3].(*Future)
93          r := newArgs[2].(Pipeline)
94          var err error
95          n := ft.NumIn() - 1
96          for _, dependencyArg := range fut.DependencyArgs {
97                  oldArgs[dependencyArg.Argument] =
                    ↪  fut.Dependencies[dependencyArg.Dependency]
98          }
99          for i, arg := range oldArgs {
100                 var variadic bool
101                 var typ reflect.Type
102                 i += 2
103                 if i >= n && ft.IsVariadic() {
104                         i = n
105                         variadic = true
106                 }
107                 typ = ft.In(i)
108                 if variadic {
109                         typ = typ.Elem()
110                 }
111                 if futID, ok := arg.(FutureID); ok && err == nil && typ
                    ↪  != futureIDType {
112                         arg, err = r.GetResult(futID)
113                 }
114                 args = append(args, arg)
115         }
116         if err == nil {
117                 err = dispatcher.ValidateCompatibleArgumentTypes(ft,
                    ↪  args)
```

```
118         }
119         return args, err
120 }
121 func (h handler) ProcessResult(fn reflect.Type, args, newArgs
    ↪  []interface{}, result []reflect.Value, panicValue interface{}) error
    ↪  {
122         var err error
123         if panicValue != nil {
124                 var ok bool
125                 if err, ok = panicValue.(error); !ok {
126                         err = fmt.Errorf("panicked with %s", panicValue)
127                 }
128         }
129         type saver interface {
130                 SaveResult(*Future, []interface{}) error
131         }
132         reg := newArgs[1].(*register)
133         r := newArgs[2].(saver)
134         fut := newArgs[3].(*Future)
135         var results []interface{}
136         var toDBId map[string]string
137         if err == nil {
138                 results = make([]interface{}, len(result)-1)
139                 toDBId, err = reg.finalize()
140         }
141         for i := range results {
142                 results[i] = result[i].Interface()
143                 if futID, ok := results[i].(FutureID); ok && err == nil {
144                         var id string
145                         if id, ok = toDBId[futID.ID]; ok {
146                                 results[i] = FutureID{
147                                         ID:     id,
148                                         Output: futID.Output,
149                                 }
150                         }
151                 }
152         }
153         if err == nil && len(reg.children) == 1 && len(results) < 2 {
154                 var ok bool
```

```
155                 var newFut Future
156                 for _, key := range reg.children {
157                         newFut = reg.futures[key]
158                         ok = fut.Name == newFut.Name
159                 }
160                 if ok {
161                         for _, arg := range results {
162                                 var future FutureID
163                                 if future, ok = arg.(FutureID); ok {
164                                         newFut, ok =
                                        ↪  reg.futures[future.ID]
165                                         if !ok {
166                                                 err = ErrFutureNotFound
167                                                 break
168                                         }
169                                 }
170                         }
171                 }
172                 if ok && err == nil && fut.Name == newFut.Name {
173                         // Perfect! Will replace the actual frame instead
                        ↪  of adding a new
174                         // children (for tail-recursion)
175                         reg.children = reg.children[:0]
176                         fut.Data = newFut.Data
177                         fut.MaxAttempts = newFut.MaxAttempts
178                         fut.Attempts = 1
179                         fut.Status = WaitingQueue
180                         fut.Delay = newFut.Delay
181                         fut.RegisteredAt = newFut.RegisteredAt
182                         fut.StartedAt = newFut.StartedAt
183                         fut.AbortedAt = newFut.AbortedAt
184                         fut.FinishedAt = newFut.FinishedAt
185                         return err
186                 }
187         }
188         if err == nil {
189                 err = r.SaveResult(fut, results)
190         }
191         return err
```

```
192 }
```

Arquivo id.go

```
1  package pipeline
2
3  import "golang.org/x/net/context"
4
5  type taskID int
6
7  const id taskID = 0
8
9  func GetID(ctx context.Context) (string, bool) {
10         val, ok := ctx.Value(id).(string)
11         return val, ok
12 }
13
14 func setID(ctx context.Context, value string) context.Context {
15         return context.WithValue(ctx, id, value)
16 }
```

Arquivo pipeline.go

```
1  package pipeline
2
3  import (
4          "bytes"
5          "reflect"
6
7          "github.com/fjorgemota/gurudamatricula/util/pool"
8
9          "github.com/fjorgemota/gurudamatricula/util/clock"
10         "github.com/fjorgemota/gurudamatricula/util/coder"
11         "github.com/fjorgemota/gurudamatricula/util/kv"
12         "github.com/fjorgemota/gurudamatricula/util/taskqueue"
13         "github.com/sirupsen/logrus"
14 )
15
16 type pipeline struct {
```

```
        store   kv.Store
        caller  taskqueue.TaskCaller
        clk     clock.Clock
        cod     coder.Coder
        options GlobalOptions
        logger  logrus.FieldLogger
        disp    Dispatcher
}

func (r *pipeline) Dispatch(task string, args ...interface{}) ([]FutureID,
↪  error) {
        var options TaskOptions
        return r.DispatchWithOptions(task, options, args...)
}

func (r *pipeline) GetTaskOutputType(futID FutureID) (reflect.Type,
↪  error) {
        return nil, ErrFutureNotFound
}

func (r *pipeline) DispatchWithOptions(task string, options TaskOptions,
↪  args ...interface{}) ([]FutureID, error) {
        buf := pool.GetBytesBuffer()
        defer pool.PutBytesBuffer(buf)
        err := r.disp.Encode(buf, task, r, args...)
        var id string
        if err == nil {
                id, err = r.store.GenerateID(r.options.Table)
        }
        if options.MaxAttempts == 0 {
                options.MaxAttempts = r.options.MaxAttempts
        }
        fut := Future{
                Name:         task,
                Data:         buf.Bytes(),
                Status:       WaitingQueue,
                MaxAttempts:  options.MaxAttempts,
                Delay:        options.Delay,
                RegisteredAt: r.clk.Now(),
```

```
53          }
54          if err == nil {
55                  r.logger.WithField("task", task).WithField("fut_id",
                    ↪  id).Debug("Saving future")
56                  err = r.store.Set(r.options.Table, id, &fut)
57          }
58          if err == nil {
59                  r.logger.WithField("task", task).WithField("fut_id",
                    ↪  id).Debug("Dispatching future")
60                  err = r.caller.CallWithOptions(r.options.WorkTaskName,
                    ↪  taskqueue.TaskOptions{Delay: fut.Delay}, id)
61          }
62          var result []FutureID
63          if err == nil {
64                  result, err = getResults(r.disp, id, task)
65          }
66          return result, err
67  }
68
69  func (r *pipeline) SaveResult(fut *Future, results []interface{}) error {
70          var buf bytes.Buffer
71          err := r.cod.EncodeTo(results, &buf)
72          if err == nil {
73                  // Set Results pointer so we can batch the write into
                    ↪  just one update
74                  // instead of two
75                  fut.Results = buf.Bytes()
76          }
77          return err
78  }
79
80  func (r *pipeline) GetStatus(futID string) (FutureStatus, error) {
81          var fut Future
82          r.logger.WithField("fut_id", futID).Debug("Getting Status")
83          var status FutureStatus
84          err := r.store.Get(r.options.Table, futID, &fut)
85          if err == nil {
86                  status = fut.Status
87          }
```

```
88          return status, err
89  }
90
91  func (r *pipeline) GetResults(futID string) ([]interface{}, error) {
92          var fut Future
93          logger := r.logger.WithField("fut_id", futID)
94          logger.Debug("Getting Results")
95          err := r.store.Get(r.options.Table, futID, &fut)
96          if err == nil && fut.Status != Done {
97                  logger.WithField("status",
                    ↪  fut.Status.String()).Error("Invalid status detected")
98                  err = ErrInvalidStatus
99          }
100         var results []interface{}
101         if err == nil {
102                 reader := bytes.NewReader(fut.Results)
103                 err = r.cod.DecodeFrom(reader, &results)
104         }
105         return results, err
106 }
107
108 func (r *pipeline) GetResult(futID FutureID) (interface{}, error) {
109 GetResultBeginning:
110         r.logger.WithField("fut_id", futID.ID).Debug("Getting Result")
111         results, err := r.GetResults(futID.ID)
112         var result interface{}
113         if err == nil {
114                 if futID.Output >= 0 && futID.Output < len(results) {
115                         result = results[futID.Output]
116                 } else {
117                         err = ErrInvalidOutputIndex
118                 }
119         }
120         if resultFutID, ok := result.(FutureID); ok {
121                 futID = resultFutID
122                 goto GetResultBeginning
123         }
124         return result, err
125 }
```

```
126
127 func (r *pipeline) DeleteFinished(futID string) error {
128         var err error
129         var zero struct{}
130         seen := make(map[string]struct{})
131         toAnalyze := []string{futID}
132         for err == nil && len(toAnalyze) > 0 {
133                 var fut Future
134                 var actual string
135                 actual, toAnalyze = toAnalyze[0], toAnalyze[1:]
136                 if _, ok := seen[actual]; ok {
137                         continue
138                 }
139                 err = r.store.Get(r.options.Table, actual, &fut)
140                 if err == nil {
141                         if fut.Status != Done && fut.Status != Aborted {
142                                 // Silently returns nil if the task
                                    ↪ didn't ended yet
143                                 return err
144                         }
145                         seen[actual] = zero
146                         if fut.Parent != "" {
147                                 toAnalyze = append(toAnalyze, fut.Parent)
148                         }
149                         for _, dependency := range fut.Dependencies {
150                                 toAnalyze = append(toAnalyze,
                                    ↪ dependency.ID)
151                         }
152                         toAnalyze = append(toAnalyze, fut.Children...)
153                         toAnalyze = append(toAnalyze, fut.Dependents...)
154                         toAnalyze = append(toAnalyze,
                            ↪ fut.DependenciesDone...)
155                 }
156         }
157         for toDelete := range seen {
158                 if err == nil {
159                         err = r.store.Del(r.options.Table, toDelete)
160                 }
161         }
```

```
162         return err
163 }
164
165 func NewPipeline(logger logrus.FieldLogger, cod coder.Coder, clk
    ↪  clock.Clock, caller taskqueue.TaskCaller, store kv.Store, dispatcher
    ↪  Dispatcher, options GlobalOptions) Pipeline {
166         cod.Register(FutureID{})
167         return &pipeline{
168                 cod:     cod,
169                 store:   store,
170                 caller:  caller,
171                 logger:  logger,
172                 options: getDefaultOptions(options),
173                 disp:    dispatcher,
174                 clk:     clk,
175         }
176 }
```

Arquivo register.go

```
1  package pipeline
2
3  import (
4          "bytes"
5          "fmt"
6          "reflect"
7          "sync"
8
9          "github.com/fjorgemota/gurudamatricula/util/clock"
10         "github.com/fjorgemota/gurudamatricula/util/kv"
11         "github.com/fjorgemota/gurudamatricula/util/pool"
12         "github.com/sirupsen/logrus"
13 )
14
15 type register struct {
16         children    []string
17         futures     map[string]Future
18         dispatcher  Dispatcher
19         store       kv.Store
20         table       string
```

```
        lock        sync.Mutex
        pipeline    Pipeline
        clk         clock.Clock
        maxAttempts int
        dispatches  int
        logger      logrus.FieldLogger
        buffers     []*bytes.Buffer
}

func (r *register) getFuture(ID string) (Future, error) {
        var err error
        fut, ok := r.futures[ID]
        if !ok {
                r.logger.WithField("looking_for", ID).Error("Future not
                ↪ found")
                err = ErrFutureNotFound
        }
        return fut, err
}

func (r *register) GetTaskOutputType(arg FutureID) (reflect.Type, error)
↪ {
        var typ reflect.Type
        var f Future
        var err error
        if err == nil {
                f, err = r.getFuture(arg.ID)
        }
        if err == nil {
                typ, err = r.dispatcher.GetType(f.Name)
        }
        if err == nil && (typ.NumOut() <= arg.Output || arg.Output < 0) {
                r.logger.WithFields(logrus.Fields{"looking_for": arg.ID,
                ↪ "name": f.Name, "output": arg.Output}).Error("Invalid
                ↪ output index")
                err = ErrInvalidOutputIndex
        }
        if err == nil {
                typ = typ.Out(arg.Output)
```

```
56          }
57          return typ, err
58  }
59
60  func (r *register) Dispatch(task string, args ...interface{}) ([]FutureID,
    ↪  error) {
61          var options TaskOptions
62          return r.DispatchWithOptions(task, options, args...)
63  }
64
65  func (r *register) dispatchChild(task string, options TaskOptions, args
    ↪  ...interface{}) ([]FutureID, error) {
66          id := fmt.Sprintf("dispatch-%s-%d", task, r.dispatches)
67          buf := pool.GetBytesBuffer()
68          err := r.dispatcher.Encode(buf, task, r, args...)
69          r.buffers = append(r.buffers, buf)
70          var f Future
71          if options.MaxAttempts == 0 {
72                  options.MaxAttempts = r.maxAttempts
73          }
74          if err == nil {
75                  f = Future{
76                          Data:         buf.Bytes(),
77                          Name:         task,
78                          MaxAttempts:  options.MaxAttempts,
79                          Delay:        options.Delay,
80                          RegisteredAt: r.clk.Now(),
81                  }
82          }
83          var futures []FutureID
84          for i, arg := range args {
85                  var isSingle bool
86                  if dependentArg, ok := arg.(FutureID); err == nil && ok {
87                          arg = []FutureID{dependentArg}
88                          isSingle = true
89                  }
90                  if dependentsArg, ok := arg.([]FutureID); err == nil &&
                    ↪  ok {
91                          if isSingle {
```

```
92                                  f.DependencyArgs =
                                    ↪  append(f.DependencyArgs,
                                    ↪  DependencyArg{
93                                          Argument:   i,
94                                          Dependency: len(futures),
95                                  })
96                          }
97                          futures = append(futures, dependentsArg...)
98                  }
99          }
100         for _, dependentArg := range futures {
101                 var dependent Future
102                 if err == nil {
103                         dependent, err = r.getFuture(dependentArg.ID)
104                 }
105                 if err == nil {
106                         f.Dependencies = append(f.Dependencies,
                                ↪  dependentArg)
107                         dependent.Dependents =
                                ↪  append(dependent.Dependents, id)
108                         r.futures[dependentArg.ID] = dependent
109                 }
110         }
111         if err == nil {
112                 r.children = append(r.children, id)
113                 r.futures[id] = f
114         }
115         var result []FutureID
116         if err == nil {
117                 result, err = getResults(r.dispatcher, id, task)
118         }
119         return result, err
120 }
121
122 func (r *register) DispatchWithOptions(task string, options TaskOptions,
    ↪  args ...interface{}) ([]FutureID, error) {
123         r.lock.Lock()
124         defer r.lock.Unlock()
125         r.dispatches++
```

```
126         if options.Detach {
127                 return r.pipeline.DispatchWithOptions(task, options,
                    ↪  args...)
128         }
129         return r.dispatchChild(task, options, args...)
130 }
131
132 func (r *register) GetResults(futID string) ([]interface{}, error) {
133         return r.pipeline.GetResults(futID)
134 }
135 func (r *register) GetResult(futID FutureID) (interface{}, error) {
136         return r.pipeline.GetResult(futID)
137 }
138
139 func (r *register) GetStatus(futID string) (FutureStatus, error) {
140         return r.pipeline.GetStatus(futID)
141 }
142
143 func (r *register) DeleteFinished(futID string) error {
144         return r.pipeline.DeleteFinished(futID)
145 }
146
147 func (r *register) finalize() (map[string]string, error) {
148         toDBId := make(map[string]string, len(r.children))
149         children := make([]string, 0, len(toDBId))
150         var err error
151         for _, child := range r.children {
152                 var newID string
153                 if err == nil {
154                         newID, err = r.store.GenerateID(r.table)
155                 }
156                 if err == nil {
157                         toDBId[child] = newID
158                         children = append(children, newID)
159                 }
160         }
161         if err == nil {
162                 r.children = children
163                 futures := make(map[string]Future, len(r.futures))
```

```
164                 for child, fut := range r.futures {
165                         dependencies := make([]FutureID, 0,
                            ↪ len(fut.Dependencies))
166                         for _, dependency := range fut.Dependencies {
167                                 dependencies = append(dependencies,
                                    ↪ FutureID{
168                                         ID:     toDBId[dependency.ID],
169                                         Output: dependency.Output,
170                                 })
171                         }
172                         dependents := make([]string, 0,
                            ↪ len(fut.Dependents))
173                         for _, dependent := range fut.Dependents {
174                                 dependents = append(dependents,
                                    ↪ toDBId[dependent])
175                         }
176                         futures[toDBId[child]] = Future{
177                                 Data:           fut.Data,
178                                 Name:           fut.Name,
179                                 MaxAttempts:    fut.MaxAttempts,
180                                 Dependencies:   dependencies,
181                                 Dependents:     dependents,
182                                 DependencyArgs: fut.DependencyArgs,
183                                 RegisteredAt:   fut.RegisteredAt,
184                                 StartedAt:      fut.StartedAt,
185                                 AbortedAt:      fut.AbortedAt,
186                                 FinishedAt:     fut.FinishedAt,
187                                 Delay:          fut.Delay,
188                         }
189                 }
190                 r.futures = futures
191         }
192         return toDBId, err
193 }
194 func newRegister(logger logrus.FieldLogger, d Dispatcher, db kv.Store,
    ↪ pipeline Pipeline, clk clock.Clock, table string, maxAttempts int)
    ↪ *register {
195         return &register{
196                 futures:     make(map[string]Future),
```

```
197                 dispatcher:  d,
198                 store:       db,
199                 table:       table,
200                 clk:         clk,
201                 logger:      logger,
202                 pipeline:    pipeline,
203                 maxAttempts: maxAttempts,
204         }
205 }
```

Arquivo result.go

```
1  package pipeline
2
3  func getResults(d Dispatcher, id, task string) ([]FutureID, error) {
4          var n int
5          typ, err := d.GetType(task)
6          if err == nil {
7                  n = typ.NumOut() - 1
8          }
9          result := make([]FutureID, 0, n)
10         for i := 0; i < n; i++ {
11                 result = append(result, FutureID{
12                         ID:     id,
13                         Output: i,
14                 })
15         }
16         return result, err
17 }
```

Arquivo worker.go

```
1  package pipeline
2
3  import (
4          "bytes"
5          "fmt"
6
7          "github.com/fjorgemota/gurudamatricula/util/clock"
```

```
8           "github.com/fjorgemota/gurudamatricula/util/coder"
9           "github.com/fjorgemota/gurudamatricula/util/kv"
10          "github.com/fjorgemota/gurudamatricula/util/pool"
11          "github.com/fjorgemota/gurudamatricula/util/taskqueue"
12          "github.com/pkg/errors"
13          "github.com/sirupsen/logrus"
14          "golang.org/x/net/context"
15  )
16
17  // WorkerHandler is an instance defining what's needed for the worker to
    ↪  worker
18  //
19  type WorkerHandler interface {
20          GetStore(context.Context) (kv.Store, error)
21          GetTaskQueueCaller(context.Context) taskqueue.TaskCaller
22          GetLogger(context.Context) logrus.FieldLogger
23  }
24  type worker struct {
25          cod        coder.Coder
26          dispatcher Dispatcher
27          handler    WorkerHandler
28          options    GlobalOptions
29          clk        clock.Clock
30  }
31
32  func (w worker) ParentDone(ctx context.Context, parentID, childID string)
    ↪  error {
33          var fut Future
34          globalDb, err := w.handler.GetStore(ctx)
35          caller := w.handler.GetTaskQueueCaller(ctx)
36          logger := w.handler.GetLogger(ctx)
37          logger = logger.WithFields(logrus.Fields{
38                  "package":   "pipeline",
39                  "action":    "ParentDone",
40                  "parent_id": parentID,
41                  "child_id":  childID,
42          })
43          var modified bool
44          if err == nil {
```

```
45                  err = globalDb.Transaction([]kv.Reference{{Table:
                    ↪  w.options.Table, Key: parentID}}, func(db
                    ↪  kv.LimitedStore) error {
46                          errParent := db.Get(w.options.Table, parentID,
                            ↪  &fut)
47                          if errParent == nil {
48                                  errParent = ErrInvalidStatus
49                          }
50                          if errParent == ErrInvalidStatus && fut.Status ==
                            ↪  WaitingChildren {
51                                  errParent = nil
52                          }
53                          if errParent == nil {
54                                  modified = true
55                                  var zero struct{}
56                                  children := make(map[string]struct{},
                                    ↪  len(fut.ChildrenWithoutDependentsDone))
57                                  for _, child := range
                                    ↪  fut.ChildrenWithoutDependentsDone {
58                                          children[child] = zero
59                                  }
60                                  if _, ok := children[childID]; !ok {
61                                          children[childID] = zero
62                                          fut.ChildrenWithoutDependentsDone
                                            ↪  =
                                            ↪  append(fut.ChildrenWithoutDependentsDon
                                            ↪  childID)
63                                          modified = true
64                                  }
65                                  if modified &&
                                    ↪  len(fut.ChildrenWithoutDependentsDone)
                                    ↪  == len(fut.ChildrenWithoutDependents)
                                    ↪  {
66                                          // It's done!
67                                          fut.Status = Done
68                                          fut.FinishedAt = w.clk.Now()
69                                  }
70                                  if modified {
```

```
71                                          errParent =
                                            ↪ db.Set(w.options.Table,
                                            ↪ parentID, &fut)
72                                  }
73                          }
74                          return errParent
75                  })
76          }
77          if fut.Status == Done && modified {
78                  for _, dependent := range fut.Dependents {
79                          if err == nil {
80                                  logger.WithField("dependent_id",
                                    ↪ dependent).Debug("Processing
                                    ↪ dependent of parent")
81                                  err = w.processDependent(logger, globalDb,
                                    ↪ caller, parentID, dependent)
82                          }
83                  }
84                  if len(fut.Parent) > 0 && err == nil &&
                    ↪ len(fut.Dependents) == 0 {
85                          // Recursions rules! \o/
86                          logger.WithField("new_parent_id",
                            ↪ fut.Parent).Debug("Calling ParentDone on new
                            ↪ parent")
87                          err = caller.Call(w.options.ParentDoneTaskName,
                            ↪ fut.Parent, parentID)
88                  }
89          }
90          return err
91  }
92
93  func (w worker) processDependent(logger logrus.FieldLogger, db kv.Store,
    ↪ caller taskqueue.TaskCaller, dependencyID, dependentID string) error
    ↪ {
94          var futDependent Future
95          var modified bool
96          err := db.Transaction([]kv.Reference{{Table: w.options.Table, Key:
            ↪ dependentID}}, func(db kv.LimitedStore) error {
```

```
97                  errDependent := db.Get(w.options.Table, dependentID,
                    ↪  &futDependent)
98                  if errDependent == nil && futDependent.Status !=
                    ↪  WaitingDependency {
99                          errDependent = ErrInvalidStatus
100                 }
101                 if errDependent == nil {
102                         var zero struct{}
103                         modified = false
104                         terminated := make(map[string]struct{},
                            ↪  len(futDependent.DependenciesDone))
105                         for _, dependency := range
                            ↪  futDependent.DependenciesDone {
106                             terminated[dependency] = zero
107                         }
108                         if _, ok := terminated[dependencyID]; !ok {
109                             terminated[dependencyID] = zero
110                             futDependent.DependenciesDone =
                                ↪  append(futDependent.DependenciesDone,
                                ↪  dependencyID)
111                             modified = true
112                         }
113                         var count int
114                         for _, dependency := range
                            ↪  futDependent.Dependencies {
115                             if _, ok := terminated[dependency.ID]; ok
                                ↪  {
116                                     count++
117                             }
118                         }
119                         if len(futDependent.Dependencies) == count {
120                             futDependent.Status = WaitingQueue
121                             modified = true
122                         }
123                         if modified {
124                             errDependent = db.Set(w.options.Table,
                                ↪  dependentID, &futDependent)
125                         }
126                 }
```

```
127                 return errDependent
128         })
129         if err == nil && futDependent.Status == WaitingQueue && modified
            ↪ {
130                 logger.WithField("dependent_id",
                    ↪ dependentID).Debug("Calling dependent")
131                 err = caller.CallWithOptions(w.options.WorkTaskName,
                    ↪ taskqueue.TaskOptions{Delay: futDependent.Delay},
                    ↪ dependentID)
132         }
133         if err == ErrInvalidStatus {
134                 // Can safely ignore ErrInvalidStatus because it just
                    ↪ means that
135                 // this task is already running
136                 err = nil
137         }
138         return err
139 }
140
141 func (w worker) dispatchChildren(caller *register, globalDb kv.Store,
    ↪ globalCaller taskqueue.TaskCaller, globalLogger logrus.FieldLogger,
    ↪ futID string, fut *Future) error {
142         var zero struct{}
143         var canBeLaunched []string
144         var err error
145         batch := globalDb.Batch()
146         seen := make(map[string]struct{}, len(caller.children))
147         seenWithoutDependents := make(map[string]struct{},
            ↪ len(caller.children))
148         for _, child := range caller.children {
149                 seen[child] = zero
150                 childFut := caller.futures[child]
151                 if len(childFut.Dependents) == 0 {
152                         seenWithoutDependents[child] = zero
153                 }
154                 var status FutureStatus
155                 if len(childFut.Dependencies) == 0 {
156                         canBeLaunched = append(canBeLaunched, child)
157                         status = WaitingQueue
```

```
158                 } else {
159                         status = WaitingDependency
160                 }
161                 reg := Future{
162                         Parent:         futID,
163                         Name:           childFut.Name,
164                         Data:           childFut.Data,
165                         Dependencies:   childFut.Dependencies,
166                         Dependents:     childFut.Dependents,
167                         Status:         status,
168                         MaxAttempts:    childFut.MaxAttempts,
169                         Delay:          childFut.Delay,
170                         RegisteredAt:   childFut.RegisteredAt,
171                         StartedAt:      childFut.StartedAt,
172                         AbortedAt:      childFut.AbortedAt,
173                         FinishedAt:     childFut.FinishedAt,
174                         DependencyArgs: childFut.DependencyArgs,
175                 }
176                 if err == nil {
177                         err = batch.Set(w.options.Table, child, &reg)
178                 }
179         }
180         fut.Children = make([]string, 0, len(seen))
181         fut.ChildrenWithoutDependents = make([]string, 0,
        ↪  len(seenWithoutDependents))
182         for child := range seen {
183                 fut.Children = append(fut.Children, child)
184         }
185         for child := range seenWithoutDependents {
186                 fut.ChildrenWithoutDependents =
                ↪  append(fut.ChildrenWithoutDependents, child)
187         }
188         if fut.Status == Running || len(seen) > 0 {
189                 if len(canBeLaunched) > 0 {
190                         fut.Status = WaitingChildren
191                 } else {
192                         fut.Status = Done
193                         fut.FinishedAt = w.clk.Now()
194                 }
```

```
195                 globalLogger.WithField("status",
                    ↪  fut.Status.String()).Debug("Updating status")
196         } else if fut.Status == WaitingQueue {
197                 // Supports tail-recursion here
198                 globalLogger.Debug("Scheduling task again for
                    ↪  tail-recursion")
199                 canBeLaunched = append(canBeLaunched, futID)
200                 caller.futures[futID] = *fut
201         }
202         if err == nil {
203                 err = batch.Set(w.options.Table, futID, fut)
204         }
205         if err == nil {
206                 err = batch.Commit()
207         }
208         for _, task := range canBeLaunched {
209                 if err == nil {
210                         if task != futID {
211                                 globalLogger.WithField("to_schedule",
                                ↪  task).Debug("Scheduling child
                                ↪  task..")
212                         }
213                         err =
                        ↪  globalCaller.CallWithOptions(w.options.WorkTaskName,
                        ↪  taskqueue.TaskOptions{Delay:
                        ↪  caller.futures[task].Delay}, task)
214                 }
215         }
216         for _, buf := range caller.buffers {
217                 pool.PutBytesBuffer(buf)
218         }
219         return err
220 }
221
222 func (w worker) dispatchAbort(logger logrus.FieldLogger, caller
    ↪  taskqueue.TaskCaller, fut Future, abortParent bool) error {
223         var err error
224         for _, dependent := range fut.Dependents {
225                 if err == nil {
```

```
226                         logger.WithField("dependent",
                            ↪  dependent).Debug("Aborting dependent!")
227                         err = caller.Call(w.options.AbortTaskName,
                            ↪  dependent, abortParent)
228                 }
229         }
230         if len(fut.Parent) > 0 && err == nil && abortParent &&
            ↪  len(fut.Dependents) == 0 {
231                 // Recursions rules! \o/
232                 err = caller.Call(w.options.AbortTaskName, fut.Parent,
                    ↪  true)
233         }
234         return err
235 }
236
237 func (w worker) setAbort(logger logrus.FieldLogger, fut *Future) {
238         logger.Debug("Aborted!")
239         fut.Status = Aborted
240         fut.AbortedAt = w.clk.Now()
241 }
242
243 func (w worker) Abort(ctx context.Context, futID string, abortParent
    ↪  bool) error {
244         var fut Future
245         logger := w.handler.GetLogger(ctx)
246         logger = logger.WithFields(logrus.Fields{
247                 "package": "pipeline",
248                 "action":  "Abort",
249                 "fut_id":  futID,
250         })
251         logger.Debug("Aborting!")
252         globalDb, err := w.handler.GetStore(ctx)
253         caller := w.handler.GetTaskQueueCaller(ctx)
254         isSet := false
255         if err == nil {
256                 err = globalDb.Transaction([]kv.Reference{{Table:
                    ↪  w.options.Table, Key: futID}}, func(db
                    ↪  kv.LimitedStore) error {
```

```
257                         errAborted := db.Get(w.options.Table, futID,
                            ↪  &fut)
258                         if errAborted == nil {
259                                 errAborted = ErrInvalidStatus
260                         }
261                         if errAborted == ErrInvalidStatus &&
                            ↪  fut.Status&(Running|WaitingChildren|WaitingDependency|Waitir
                            ↪  > 0 {
262                                 w.setAbort(logger, &fut)
263                                 isSet = true
264                                 errAborted = db.Set(w.options.Table,
                                    ↪  futID, &fut)
265                         }
266                         return errAborted
267                 })
268         }
269         if err == nil && fut.Status == Aborted {
270                 err = w.dispatchAbort(logger, caller, fut, abortParent)
271         }
272         if err == ErrInvalidStatus {
273                 logger.WithField("status", fut.Status).Error("Wrong
                    ↪  status detected")
274         }
275         if err != nil {
276                 logger.WithError(err).Error("Error while aborting")
277         }
278         if fut.Status == Aborted && !isSet {
279                 logger.Warn("Task was already aborted..Ignoring error")
280                 err = nil
281         }
282         return err
283 }
284
285 func (w worker) Work(ctx context.Context, futID string) error {
286         globalDb, err := w.handler.GetStore(ctx)
287         globalCaller := w.handler.GetTaskQueueCaller(ctx)
288         globalLogger := w.handler.GetLogger(ctx)
```

```
289         globalLogger = globalLogger.WithField("package",
            ↪  "pipeline").WithField("action", "Work").WithField("fut_id",
            ↪  futID)
290         var fut Future
291         futLogger := globalLogger
292         shouldRetry := true
293         if err == nil {
294                 err = globalDb.Transaction([]kv.Reference{{Table:
                    ↪  w.options.Table, Key: futID}}, func(db
                    ↪  kv.LimitedStore) error {
295                         errOp := db.Get(w.options.Table, futID, &fut)
296                         if errOp == nil {
297                                 errOp = ErrInvalidStatus
298                                 futLogger =
                                    ↪  globalLogger.WithField("work_name",
                                    ↪  fut.Name).WithField("work_parent",
                                    ↪  fut.Parent)
299                         }
300                         if errOp == ErrInvalidStatus && fut.Status ==
                            ↪  WaitingQueue {
301                                 futLogger.WithField("from",
                                    ↪  fut.Status).Debug("Changing status to
                                    ↪  Running")
302                                 errOp = nil
303                         }
304                         fut.Status = Running
305                         fut.StartedAt = w.clk.Now()
306                         fut.Attempts++
307                         if errOp == nil {
308                                 futLogger.WithField("attempts",
                                    ↪  fut.Attempts).Debug("Saving Future as
                                    ↪  Running")
309                                 errOp = db.Set(w.options.Table, futID,
                                    ↪  &fut)
310                         }
311                         return errOp
312                 })
313         }
314         var caller *register
```

```
315         if err == nil {
316                 // Execute the function correctly!
317                 pipeline := NewPipeline(futLogger, w.cod, w.clk,
                    ↪  globalCaller, globalDb, w.dispatcher, w.options)
318                 caller = newRegister(futLogger, w.dispatcher, globalDb,
                    ↪  pipeline, w.clk, w.options.Table, fut.MaxAttempts)
319                 shouldRetry, err = w.dispatcher.Decode(setID(ctx, futID),
                    ↪  caller, pipeline, &fut, bytes.NewReader(fut.Data))
320         }
321         if err == nil {
322                 shouldRetry = true
323         }
324         if err == nil {
325                 err = w.dispatchChildren(caller, globalDb, globalCaller,
                    ↪  futLogger, futID, &fut)
326                 caller = nil
327         }
328         if err == nil && fut.Status == Done {
329                 futLogger.Debug("Task completed successfully")
330                 // Can fan out all the updates! \o/
331                 // Needs to update parent (recursively, because parent
                    ↪  may have dependents or other parents)
332                 if len(fut.Parent) > 0 && len(fut.Dependents) == 0 {
333                         futLogger.Debug("Configuring Parent as Done")
334                         err =
                            ↪  globalCaller.Call(w.options.ParentDoneTaskName,
                            ↪  fut.Parent, futID)
335                 }
336                 for _, dependent := range fut.Dependents {
337                         if err == nil {
338                                 futLogger.WithField("dependent",
                                    ↪  dependent).Debug("Dispatching
                                    ↪  information to dependent")
339                                 err = w.processDependent(globalLogger,
                                    ↪  globalDb, globalCaller, futID,
                                    ↪  dependent)
340                         }
341                 }
342         }
```

```
343         if err != nil {
344                 // Should Retry should be managed here!
345                 errorLogger :=
                    ↪  futLogger.WithError(err).WithField("error_stack",
                    ↪  getStack(err)).WithField("error_message",
                    ↪  err.Error())
346                 if shouldRetry && fut.Attempts < fut.MaxAttempts {
347                         // Save the future and retry again manually!
348                         errorLogger.Warn("Scheduling task again after
                            ↪  error")
349                         fut.Status = WaitingQueue
350                         err = globalDb.Set(w.options.Table, futID, &fut)
351                         if err == nil {
352                                 // If the Call below causes an error too,
                                    ↪  the task queue will try
353                                 // to repeat the task again
354                                 err =
                                    ↪  globalCaller.Call(w.options.WorkTaskName,
                                    ↪  futID)
355                         }
356                 } else {
357                         // Check if error was caused by corruption in the
                            ↪  function call(?) and
358                         // if so abort all the chain of processes
359                         msg := "Aborting task after corruption error"
360                         if fut.Attempts >= fut.MaxAttempts {
361                                 errorLogger =
                                    ↪  errorLogger.WithField("attempts",
                                    ↪  fut.Attempts).WithField("max_attempts",
                                    ↪  fut.MaxAttempts)
362                                 msg = "Aborting task after maximum number
                                    ↪  of attempts reached"
363                         }
364                         errorLogger.Warn(msg)
365                         w.setAbort(globalLogger, &fut)
366                         err = globalDb.Set(w.options.Table, futID, &fut)
367                         if err == nil {
368                                 err = w.dispatchAbort(globalLogger,
                                    ↪  globalCaller, fut, true)
```

```
                        }
                }
        }
        return err
}

func getStack(err error) string {
        type causer interface {
                Cause() error
        }
        type stackTracer interface {
                StackTrace() errors.StackTrace
        }
        var result string
        var cerr causer
start:
        _, ok := err.(stackTracer)
        if ok {
                cerr, ok = err.(causer)
                if ok {
                        // Will add a new stack frame here so we can
                        ↪  detect if there is a
                        // deeper stack
                        result = getStack(cerr.Cause())
                }
                if len(result) == 0 {
                        result = fmt.Sprintf("%+v", err)
                }
        } else {
                cerr, ok = err.(causer)
                if ok {
                        err = cerr.Cause()
                        goto start
                }
        }
        return result
}

```

```
406 func RegisterWorker(taskDispatcher taskqueue.TaskDispatcher, dispatcher
    ↪  Dispatcher, cod coder.Coder, clk clock.Clock, handler WorkerHandler,
    ↪  options GlobalOptions) error {
407         options = getDefaultOptions(options)
408         cod.Register(FutureID{})
409         w := worker{
410                 options:    options,
411                 dispatcher: dispatcher,
412                 handler:    handler,
413                 cod:        cod,
414                 clk:        clk,
415         }
416         err := taskDispatcher.Register(options.WorkTaskName, w.Work)
417         if err == nil {
418                 err = taskDispatcher.Register(options.ParentDoneTaskName,
                    ↪  w.ParentDone)
419         }
420         if err == nil {
421                 err = taskDispatcher.Register(options.AbortTaskName,
                    ↪  w.Abort)
422         }
423         return err
424 }
```

B.1.31 Pasta util/pool

Arquivo atoi.go

```
1  package pool
2
3  import (
4          "fmt"
5          "math"
6          "sync"
7
8          "github.com/pkg/errors"
9  )
10
11 type atoiError struct {
12         b byte
```

```
13 }
14
15 func (aerr atoiError) Error() string {
16         return fmt.Sprintf("pool: Number not recognized: '%c'", aerr.b)
17 }
18
19 func atoi(bs []byte) (int, error) {
20         var r int
21         l := float64(len(bs)) - 1
22         var err error
23         if len(bs) == 0 {
24                 err = errors.New("pool: Cannot recognize number from
                   ↪  empty byte slice")
25         }
26         mult := 1
27         for i, n := range bs {
28                 if i == 0 && n == '-' {
29                         mult = -1
30                         l--
31                 } else if n >= '0' && n <= '9' {
32                         r += int(math.Pow(10, l)) * int(n-'0') * mult
33                         l--
34                 } else {
35                         err = errors.WithStack(atoiError{n})
36                         break
37                 }
38         }
39         return r, err
40 }
41
42 type AtoiCache interface {
43         Atoi(content []byte) (int, error)
44 }
45 type atoiCache struct {
46         cache map[string]int
47 }
48
49 func (c *atoiCache) Atoi(content []byte) (int, error) {
50         var value int
```

```
51          var ok bool
52          var err error
53          if value, ok = c.cache[string(content)]; !ok {
54                  value, err = atoi(content)
55                  if err == nil {
56                          c.cache[string(content)] = value
57                  }
58          }
59          return value, err
60  }
61
62  func newAtoiCache() AtoiCache {
63          return &atoiCache{
64                  cache: make(map[string]int, 256),
65          }
66  }
67
68  var atoiPool = sync.Pool{
69          New: func() interface{} {
70                  return newAtoiCache()
71          },
72  }
73
74  func GetAtoiCache() AtoiCache {
75          return atoiPool.Get().(AtoiCache)
76  }
77
78  func PutAtoiCache(cache AtoiCache) {
79          atoiPool.Put(cache)
80  }
```

Arquivo bufio.go

```
1   package pool
2
3   import (
4           "bufio"
5           "io"
6           "sync"
7   )
```

```
8
9  var bufioReaderPool = sync.Pool{
10         New: func() interface{} {
11                 return bufio.NewReader(nil)
12         },
13 }
14
15 var bufioWriterPool = sync.Pool{
16         New: func() interface{} {
17                 return bufio.NewWriter(nil)
18         },
19 }
20
21 // GetBufioReader returns a bufio.Reader ready to use
22 func GetBufioReader(reader io.Reader) *bufio.Reader {
23         result := bufioReaderPool.Get().(*bufio.Reader)
24         result.Reset(reader)
25         return result
26 }
27
28 // PutBufioReader allows a bufio.Reader to be reused
29 func PutBufioReader(buf *bufio.Reader) {
30         bufioReaderPool.Put(buf)
31 }
32
33 // GetBufioWriter returns a bufio.Writer ready to use
34 func GetBufioWriter(writer io.Writer) *bufio.Writer {
35         result := bufioWriterPool.Get().(*bufio.Writer)
36         result.Reset(writer)
37         return result
38 }
39
40 // PutBufioWriter allows a bufio.Writer to be reused
41 func PutBufioWriter(buf *bufio.Writer) {
42         bufioWriterPool.Put(buf)
43 }
```

Arquivo bytes.go

```
1  package pool
```

```
2
3  import (
4          "bytes"
5          "sync"
6  )
7
8  var bytesPool = sync.Pool{
9          New: func() interface{} {
10                 return new(bytes.Buffer)
11         },
12 }
13
14 // GetBytesBuffer return a bytes.Buffer ready to use
15 func GetBytesBuffer() *bytes.Buffer {
16         return bytesPool.Get().(*bytes.Buffer)
17 }
18
19 // PutBytesBuffer puts a bytes.Buffer to reuse
20 func PutBytesBuffer(buf *bytes.Buffer) {
21         if buf == nil {
22                 return
23         }
24         buf.Reset()
25         bytesPool.Put(buf)
26 }
```

Arquivo sha1.go

```
1  package pool
2
3  import (
4          "crypto/sha1"
5          "hash"
6          "sync"
7  )
8
9  var sha1Pool = sync.Pool{
10         New: func() interface{} {
11                 return sha1.New()
12         },
```

```
13 }
14
15 func GetSha1() hash.Hash {
16         return sha1Pool.Get().(hash.Hash)
17 }
18
19 func PutSha1(h hash.Hash) {
20         h.Reset()
21         sha1Pool.Put(h)
22 }
```

Arquivo string.go

```
1  package pool
2
3  import "sync"
4
5  type StringCache interface {
6          String(data []byte) string
7          Transform(data []byte, transform func(data []byte) string) string
8  }
9
10 type stringCache struct {
11         cache map[string]string
12 }
13
14 func (tc *stringCache) String(data []byte) string {
15         return tc.Transform(data, func(data []byte) string {
16                 return string(data)
17         })
18 }
19
20 func (tc *stringCache) Transform(data []byte, transform func(data []byte)
   ↪  string) string {
21         var result string
22         var ok bool
23         if result, ok = tc.cache[string(data)]; !ok {
24                 result = transform(data)
25                 tc.cache[string(data)] = result
26         }
```

```
27          return result
28  }
29
30  func newStringCache() StringCache {
31          return &stringCache{cache: make(map[string]string, 256)}
32  }
33
34  var stringCachePool = sync.Pool{
35          New: func() interface{} {
36                  return newStringCache()
37          },
38  }
39
40  func GetStringCache() StringCache {
41          return stringCachePool.Get().(StringCache)
42  }
43
44  func PutStringCache(cache StringCache) {
45          stringCachePool.Put(cache)
46  }
```

B.1.32 Pasta util/taskqueue

Arquivo appengine.go

```
1   package taskqueue
2
3   import (
4           "github.com/fjorgemota/gurudamatricula/util/pool"
5           "golang.org/x/net/context"
6
7           "google.golang.org/appengine/taskqueue"
8   )
9
10  type AppEngineOptions struct {
11          Base      string
12          QueueName string
13          Retries   int
14  }
15
```

```
16 type appEngineCaller struct {
17         ctx     context.Context
18         d       TaskDispatcher
19         options AppEngineOptions
20 }
21
22 func (aec *appEngineCaller) Call(name string, args ...interface{}) error
   ↪ {
23         return aec.CallWithOptions(name, TaskOptions{}, args...)
24 }
25
26 func (aec *appEngineCaller) CallWithOptions(name string, options
   ↪  TaskOptions, args ...interface{}) error {
27         buf := pool.GetBytesBuffer()
28         defer pool.PutBytesBuffer(buf)
29         err := aec.d.Encode(buf, name, args...)
30         if err == nil {
31                 if options.MaxRetries <= 0 {
32                         options.MaxRetries = aec.options.Retries
33                 }
34                 task := taskqueue.Task{
35                         Payload: buf.Bytes(),
36                         Path:    aec.options.Base,
37                         Method:  "POST",
38                         RetryOptions: &taskqueue.RetryOptions{
39                                 RetryLimit: int32(options.MaxRetries),
40                         },
41                         Delay: options.Delay,
42                 }
43                 _, err = taskqueue.Add(aec.ctx, &task,
                   ↪  aec.options.QueueName)
44         }
45         return err
46 }
47
48 func (aec *appEngineCaller) Close() error {
49         return nil
50 }
51
```

```
52 // NewAppEngineCaller returns a caller that uses AppEngine's API to make
   ↪  the calls
53 func NewAppEngineCaller(ctx context.Context, d TaskDispatcher, options
   ↪  AppEngineOptions) TaskCaller {
54         return &appEngineCaller{
55                 options: options,
56                 ctx:     ctx,
57                 d:       d,
58         }
59 }
```

Arquivo base.go

```
1  package taskqueue
2
3  import (
4          "errors"
5          "io"
6          "net/http"
7          "reflect"
8          "time"
9
10         "golang.org/x/net/context"
11 )
12
13 var (
14         // precomputed types
15         contextType = reflect.TypeOf((*context.Context)(nil)).Elem()
16 )
17
18 var (
19         // ErrClosed is returned when the task queue was already closed
20         ErrClosed = errors.New("TaskQueue already closed")
21 )
22
23 // HTTPRouterHandler implements the Handle API of httprouter package
24 type HTTPRouterHandler interface {
25         Handler(method, path string, handler http.Handler)
26 }
27
```

```
28 // MuxHandler implements the Handle API of http's ServerMux
29 type MuxHandler interface {
30         Handle(pattern string, handler http.Handler)
31 }
32
33 // TaskDispatcher is a interface where we can register tasks easily
34 type TaskDispatcher interface {
35         Register(name string, fn interface{}) error
36         Decode(ctx context.Context, reader io.Reader) (bool, error)
37         Encode(writer io.Writer, name string, args ...interface{}) error
38 }
39
40 // TaskOptions is a struct where you can pass additional instructions for
   ↪  the
41 // task dispatcher
42 type TaskOptions struct {
43         Delay      time.Duration
44         MaxRetries int
45 }
46
47 // TaskCaller calls tasks based on a TaskDispatcher
48 type TaskCaller interface {
49         Call(name string, args ...interface{}) error
50         CallWithOptions(name string, options TaskOptions, args
          ↪  ...interface{}) error
51         Close() error
52 }
```

Arquivo caller.go

```
1  package taskqueue
2
3  import (
4          "bytes"
5          "io"
6          "io/ioutil"
7          "net/http"
8          "runtime"
9          "sync"
10
```

```
11          "github.com/fjorgemota/gurudamatricula/util/clock"
12          "github.com/fjorgemota/gurudamatricula/util/pool"
13          "github.com/sirupsen/logrus"
14  )
15
16  type inlineCaller struct {
17          base    string
18          retries int
19          client  *http.Client
20          logger  logrus.FieldLogger
21          queue   chan int
22          d       TaskDispatcher
23          clk     clock.Clock
24          wg      sync.WaitGroup
25  }
26
27  func (ic *inlineCaller) call(buf *bytes.Buffer, options TaskOptions,
    ↪  logger logrus.FieldLogger) {
28          defer pool.PutBytesBuffer(buf)
29          defer ic.wg.Done()
30          logger.Debugf("Sleeping for %s", options.Delay)
31          ic.clk.Sleep(options.Delay)
32          var value int
33          if ic.queue != nil {
34                  value = <-ic.queue
35          }
36          for c := 0; c < options.MaxRetries; c++ {
37                  log := logger.WithField("attempt", c)
38                  log.Debugf("Calling task..")
39                  request, err := http.NewRequest("POST", ic.base,
                    ↪  bytes.NewReader(buf.Bytes()))
40                  var response *http.Response
41                  if err == nil {
42                          request.Close = true
43                          response, err = ic.client.Do(request)
44                  }
45                  if err == nil {
46                          _, err = io.Copy(ioutil.Discard, response.Body)
47                  }
```

```
48                  if err == nil {
49                          err = response.Body.Close()
50                  }
51                  if err == nil && response.StatusCode >= 200 &&
                    ↪  response.StatusCode < 300 {
52                          log.Debug("Task executed successfully! Stopping
                            ↪  attempts now..")
53                          break
54                  } else {
55                          if response != nil {
56                                  log = log.WithField("status_code",
                                    ↪  response.StatusCode)
57                          }
58                          log.WithError(err).Errorf("Error ocurred while
                            ↪  calling task")
59                  }
60          }
61          if ic.queue != nil {
62                  ic.queue <- value
63          }
64  }
65
66  func (ic *inlineCaller) Call(name string, args ...interface{}) error {
67          return ic.CallWithOptions(name, TaskOptions{}, args...)
68  }
69
70  func (ic *inlineCaller) CallWithOptions(name string, options TaskOptions,
    ↪  args ...interface{}) error {
71          buf := pool.GetBytesBuffer()
72          err := ic.d.Encode(buf, name, args...)
73          if err == nil {
74                  if options.MaxRetries < 1 {
75                          options.MaxRetries = ic.retries
76                  }
77                  ic.wg.Add(1)
78                  go ic.call(buf, options, ic.logger)
79          }
80          return err
81  }
```

```
82
83  func (ic *inlineCaller) Close() error {
84          ic.wg.Wait()
85          return nil
86  }
87
88  type InlineCallerOptions struct {
89          Base        string
90          Retries     int
91          Concurrency int
92          Client      *http.Client
93          Logger      logrus.FieldLogger
94          Clock       clock.Clock
95  }
96
97  func NewInlineCaller(d TaskDispatcher, options InlineCallerOptions)
    ↪  TaskCaller {
98          if options.Client == nil {
99                  options.Client = http.DefaultClient
100         }
101         if options.Clock == nil {
102                 options.Clock = clock.NewRealClock()
103         }
104         if options.Logger == nil {
105                 log := logrus.New()
106                 log.Out = ioutil.Discard
107                 options.Logger = log
108         }
109         var queue chan int
110         if options.Concurrency == 0 {
111                 options.Concurrency = runtime.NumCPU()
112         }
113         if options.Retries < 1 {
114                 options.Retries = 1
115         }
116         if options.Concurrency > 0 {
117                 queue = make(chan int, options.Concurrency)
118                 for c := 0; c < options.Concurrency; c++ {
119                         queue <- c + 1
```

```
120                 }
121         }
122         base := options.Base
123         return &inlineCaller{
124                 base:    base,
125                 retries: options.Retries,
126                 client:  options.Client,
127                 logger:  options.Logger.WithFields(logrus.Fields{"base":
                    ↪  base, "retries": options.Retries, "package":
                    ↪  "pipeline"}),
128                 d:       d,
129                 queue:   queue,
130                 clk:     options.Clock,
131                 wg:      sync.WaitGroup{},
132         }
133 }
```

Arquivo cloudtasks.go

```
1  package taskqueue
2
3  import (
4          "net/http"
5          "strconv"
6
7          cloudtasks "cloud.google.com/go/cloudtasks/apiv2"
8          "github.com/fjorgemota/gurudamatricula/util/clock"
9          "github.com/fjorgemota/gurudamatricula/util/pool"
10         "github.com/golang/protobuf/ptypes/timestamp"
11         "golang.org/x/net/context"
12         "google.golang.org/genproto/googleapis/cloud/tasks/v2"
13 )
14
15 type CloudTasksOptions struct {
16         Base       string
17         ProjectID  string
18         LocationID string
19         QueueID    string
20         Retries    int
21 }
```

```
22
23 type cloudTasksCaller struct {
24         ctx     context.Context
25         d       TaskDispatcher
26         clk     clock.Clock
27         options CloudTasksOptions
28         client  *cloudtasks.Client
29 }
30
31 func (ctc *cloudTasksCaller) Call(name string, args ...interface{}) error
   ↪ {
32         return ctc.CallWithOptions(name, TaskOptions{}, args...)
33 }
34
35 func (ctc *cloudTasksCaller) CallWithOptions(name string, options
   ↪  TaskOptions, args ...interface{}) error {
36         buf := pool.GetBytesBuffer()
37         defer pool.PutBytesBuffer(buf)
38         err := ctc.d.Encode(buf, name, args...)
39         if err == nil {
40                 if options.MaxRetries <= 0 {
41                         options.MaxRetries = ctc.options.Retries
42                 }
43                 task := &tasks.CreateTaskRequest{
44                         Parent: "projects/" + ctc.options.ProjectID +
                           ↪  "/locations/" + ctc.options.LocationID +
                           ↪  "/queues/" + ctc.options.QueueID,
45                         Task: &tasks.Task{
46                                 MessageType:
                                   ↪  &tasks.Task_AppEngineHttpRequest{
47                                         AppEngineHttpRequest:
                                           ↪  &tasks.AppEngineHttpRequest{
48                                                 Body:        buf.Bytes(),
49                                                 RelativeUri:
                                                   ↪  ctc.options.Base,
50                                                 HttpMethod:
                                                   ↪  tasks.HttpMethod_POST,
51                                                 Headers:
                                                   ↪  map[string]string{
```

```
52                                                      "X-App-MaxExecutions":
                                                        ↪ strconv.Itoa(options.MaxRetr
53                                                  },
54                                          },
55                                  },
56                                  ScheduleTime: &timestamp.Timestamp{
57                                          Seconds:
                                            ↪ ctc.clk.Now().Add(options.Delay).Unix(),
58                                  },
59                          },
60                  }
61                  _, err = ctc.client.CreateTask(ctc.ctx, task)
62          }
63          return err
64  }
65
66  func (ctc *cloudTasksCaller) Close() error {
67          return ctc.client.Close()
68  }
69
70  // NewCloudTasksCaller returns a caller that uses CloudTasks's API to
    ↪ make the calls
71  func NewCloudTasksCaller(ctx context.Context, d TaskDispatcher, options
    ↪ CloudTasksOptions, client *cloudtasks.Client) TaskCaller {
72          return &cloudTasksCaller{
73                  options: options,
74                  ctx:     ctx,
75                  d:       d,
76                  client:  client,
77          }
78  }
79
80  func CloudTasksWorker(next http.Handler) http.Handler {
81          return http.HandlerFunc(func(resp http.ResponseWriter, req
            ↪ *http.Request) {
82                  var totalExecutions int
83                  maxExecutions, err :=
                    ↪ strconv.Atoi(req.Header.Get("X-App-MaxExecutions"))
84                  if err == nil {
```

```
85                          totalExecutions, err =
                            ↪  strconv.Atoi(req.Header.Get("X-AppEngine-TaskExecutionC
86                  }
87                  if err == nil && totalExecutions <= maxExecutions {
88                          next.ServeHTTP(resp, req)
89                  }
90          })
91  }
```

Arquivo dispatcher.go

```
1   package taskqueue
2
3   import (
4           "io"
5
6           "golang.org/x/net/context"
7
8           "github.com/fjorgemota/gurudamatricula/util/coder"
9           "github.com/fjorgemota/gurudamatricula/util/dispatcher"
10  )
11
12  type taskDispatcher struct {
13          d dispatcher.Dispatcher
14  }
15
16  func (fd *taskDispatcher) Register(name string, fn interface{}) error {
17          return fd.d.Register(name, fn)
18  }
19
20  func (fd *taskDispatcher) Encode(writer io.Writer, name string, args
    ↪  ...interface{}) error {
21          return fd.d.Encode(writer, name, args...)
22  }
23
24  func (fd *taskDispatcher) Decode(ctx context.Context, reader io.Reader)
    ↪  (shouldRetry bool, err error) {
25          return fd.d.Decode(reader, ctx)
26  }
27
```

```
28 // NewDispatcher returns a new task dispatcher
29 func NewDispatcher(cod coder.Coder) TaskDispatcher {
30         return &taskDispatcher{
31                 d: dispatcher.NewDispatcher(cod, TaskHandler{}),
32         }
33 }
```

Arquivo handler.go

```
1  package taskqueue
2
3  import (
4          "errors"
5          "fmt"
6          "reflect"
7
8          "github.com/fjorgemota/gurudamatricula/util/dispatcher"
9  )
10
11 type TaskHandler struct{}
12
13 // ValidateFunction checks if functions have a conntext as first argument
14 func (h TaskHandler) ValidateFunction(ft reflect.Type) error {
15         if ft.NumIn() == 0 || ft.In(0) != contextType {
16                 return errors.New("first argument of fn should be a
                   ↪  context")
17         }
18         return nil
19 }
20
21 // ProcessCall validates if a function is valid by checking the number
   ↪  and
22 // the types of the arguments
23 func (h TaskHandler) ProcessCall(ft reflect.Type, args []interface{})
   ↪  ([]interface{}, error) {
24         nArgs := len(args) + 1
25         err := dispatcher.ValidateMinArguments(ft, nArgs)
26         if err == nil {
27                 err = dispatcher.ValidateMaxArguments(ft, nArgs)
28         }
```

```
29          var fnArgs []reflect.Type
30          if err == nil {
31                  fnArgs, err = dispatcher.GetArgumentTypes(ft)
32          }
33          if err == nil {
34                  fnArgs = fnArgs[1:]
35          }
36          if err == nil && ft.IsVariadic() && nArgs >= len(fnArgs)-1 {
37                  fnArgs = dispatcher.AdaptVariadic(fnArgs, len(args))
38          }
39          if err == nil {
40                  err = dispatcher.ValidateCompatibleTypes(fnArgs, args)
41          }
42          return args, err
43  }
44
45  // ProcessArgs just return the old arguments slice to the user
46  func (h TaskHandler) ProcessArgs(ft reflect.Type, oldArgs []interface{},
    ↪  newArgs []interface{}) ([]interface{}, error) {
47          // We assume that newArgs is always filled with ONE element of a
            ↪  context.Context
48          // just because taskDispatcher wraps this method so we do not
            ↪  need to check it
49          // again
50          args := make([]interface{}, 0, len(oldArgs)+1)
51          args = append(args, newArgs[0])
52          args = append(args, oldArgs...)
53          return args, nil
54  }
55
56  // ProcessResult is a no-op that just returns nil
57  func (h TaskHandler) ProcessResult(fn reflect.Type, args, newArgs
    ↪  []interface{}, result []reflect.Value, panicValue interface{}) error
    ↪  {
58          var err error
59          if panicValue != nil {
60                  var ok bool
61                  if err, ok = panicValue.(error); !ok {
62                          err = fmt.Errorf("panicked with %s", panicValue)
```

```
63                  }
64          }
65          return err
66  }
67
68  var _ dispatcher.Handler = TaskHandler{}
```

Arquivo worker.go

```
1   package taskqueue
2
3   import (
4           "net/http"
5
6           "github.com/sirupsen/logrus"
7
8           "golang.org/x/net/context"
9   )
10
11  type WorkerHandler interface {
12          GetContext(*http.Request) context.Context
13          GetLogger(context.Context) logrus.FieldLogger
14  }
15
16  type httpWorker struct {
17          d       TaskDispatcher
18          handler WorkerHandler
19  }
20
21  func (w httpWorker) ServeHTTP(resp http.ResponseWriter, req
    ↪  *http.Request) {
22          c := w.handler.GetContext(req)
23          shouldRetry, err := w.d.Decode(c, req.Body)
24          if err != nil {
25                  logger := w.handler.GetLogger(c)
26                  logger.Error(err)
27                  if shouldRetry {
28                          resp.WriteHeader(http.StatusInternalServerError)
29                          logger.Warning("will retry")
30                          return
```

```
                }
                logger.Warning("dropping task")
        }
        resp.WriteHeader(http.StatusOK)
}

// NewHTTPWorker creates a generic worker, that do not knows about
// any specific routers (but is compatible with httprouter package)
func NewHTTPWorker(d TaskDispatcher, handler WorkerHandler) http.Handler
↪ {
        return httpWorker{
                d:       d,
                handler: handler,
        }
}
```

B.1.33 Pasta util/testutil

Arquivo buf_closer.go

```
package testutil

import "bytes"

type BufferCloser struct {
        bytes.Buffer
}

func (bc BufferCloser) Close() error {
        return nil
}
```

Arquivo callbacker.go

```
package testutil

import "net/http"

// Callbacker is a http.RoundTripper that is defined by a function
type Callbacker func(req *http.Request) (*http.Response, error)
```

```
7
8  // RoundTrip implements http.RoundTripper interface
9  func (c Callbacker) RoundTrip(req *http.Request) (*http.Response, error)
   ↪ {
10         return c(req)
11 }
```

Arquivo comparer.go

```
1  package testutil
2
3  import (
4          "context"
5          "strings"
6
7          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
8          "github.com/fjorgemota/gurudamatricula/util/pipeline"
9          . "github.com/onsi/ginkgo"
10         . "github.com/onsi/gomega"
11 )
12
13 type ComparerVerifier interface {
14         CheckEmpty(value []string)
15         CheckSucceed(err error)
16         Recover()
17         Done()
18 }
19
20 type GinkgoVerifier struct {
21         done Done
22 }
23
24 func (gv GinkgoVerifier) CheckEmpty(value []string) {
25         Expect(value).To(BeEmpty())
26 }
27 func (gv GinkgoVerifier) CheckSucceed(err error) {
28         Expect(err).To(Succeed())
29 }
30 func (gv GinkgoVerifier) Recover() {
31         GinkgoRecover()
```

```
32 }
33 func (gv GinkgoVerifier) Done() {
34         close(gv.done)
35 }
36
37 func doneFunc(differ ddiffer.PipelineHandler, verifier ComparerVerifier)
   ↪  func(ctx context.Context, pipe pipeline.Pipeline, results
   ↪  ...ddiffer.ComparerResult) error {
38         return func(ctx context.Context, pipe pipeline.Pipeline, results
           ↪  ...ddiffer.ComparerResult) error {
39                 defer verifier.Done()
40                 defer verifier.Recover()
41                 for _, result := range results {
42                         verifier.CheckEmpty(result.Created)
43                         verifier.CheckEmpty(result.Modified)
44                         verifier.CheckEmpty(result.Deleted)
45                 }
46                 return nil
47         }
48 }
49 func compareFunc(differ ddiffer.PipelineHandler, verifier
   ↪  ComparerVerifier, universityID string, source ddiffer.SourceType)
   ↪  func(ctx context.Context, pipe pipeline.Pipeline, results
   ↪  []ddiffer.PipelineConverterResult) error {
50         return func(ctx context.Context, pipe pipeline.Pipeline, results
           ↪  []ddiffer.PipelineConverterResult) error {
51                 defer verifier.Recover()
52                 var final []interface{}
53                 var err error
54                 for _, result := range results {
55                         var comparer pipeline.FutureID
56                         if err == nil {
57                                 comparer, err = differ.Compare(pipe,
                                   ↪  ddiffer.ComparerOptions{
58                                         NewSource:        result.Result,
59                                         Target:           result.Target,
```

```
60                                      OldSource:        universityID +
                                        ↪  "-" +
                                        ↪  strings.ToLower(source.String())
                                        ↪  + "-" +
                                        ↪  strings.ToLower(result.Target.String())
                                        ↪  + ".dat",
61                                      CreatedTemplate:  "created-" +
                                        ↪  strings.ToLower(result.Target.String())
                                        ↪  + "-%d.dat",
62                                      ModifiedTemplate: "modified-" +
                                        ↪  strings.ToLower(result.Target.String())
                                        ↪  + "-%d.dat",
63                                      DeletedTemplate:  "deleted-" +
                                        ↪  strings.ToLower(result.Target.String())
                                        ↪  + "-%d.dat",
64                                  })
65                          }
66                          if err == nil {
67                                  final = append(final, comparer)
68                          }
69                  }
70                  if err == nil {
71                          _, err = pipe.Dispatch("test.comparer.done",
                            ↪  final...)
72                  }
73                  verifier.CheckSucceed(err)
74                  return err
75          }
76  }
77
78  func convertFunc(differ ddiffer.PipelineHandler, verifier
    ↪  ComparerVerifier, universityID string, source ddiffer.SourceType)
    ↪  func(ctx context.Context, pipe pipeline.Pipeline, filename string)
    ↪  error {
79          return func(ctx context.Context, pipe pipeline.Pipeline, filename
            ↪  string) error {
80                  defer verifier.Recover()
81                  result, err := differ.Convert(pipe,
                    ↪  ddiffer.PipelineConverterOptions{
```

```
82                          UniversityID:   universityID,
83                          Source:         source,
84                          Name:           filename,
85                          TargetTemplate: "result-%s.dat",
86                  })
87                  if err == nil {
88                          _, err = pipe.Dispatch("test.comparer.compare",
                            ↪  result)
89                  }
90                  verifier.CheckSucceed(err)
91                  return err
92          }
93  }
94
95  // Register tasks using Ginkgo and Gomega to verify for errors
96  func RegisterComparerFunc(dispatcher pipeline.Dispatcher, handler
    ↪  ddiffer.PipelineDependencies, done Done, universityID string, source
    ↪  ddiffer.SourceType) {
97          RegisterComparerFuncWithVerifier(dispatcher, handler,
            ↪  GinkgoVerifier{done: done}, universityID, source)
98  }
99
100 // Register a few tasks on the pipeline that will allow to convert and
    ↪  compare easily a dataset with the
101 // reference dataset, allowing to detect differences between them in a
    ↪  parallel way
102 func RegisterComparerFuncWithVerifier(dispatcher pipeline.Dispatcher,
    ↪  handler ddiffer.PipelineDependencies, verifier ComparerVerifier,
    ↪  universityID string, source ddiffer.SourceType) {
103         differ, err := ddiffer.NewPipelineHandler(dispatcher, handler,
            ↪  ddiffer.PipelineGlobalOptions{})
104         if err == nil {
105                 err = dispatcher.Register("test.comparer.done",
                    ↪  doneFunc(differ, verifier))
106         }
107         if err == nil {
108                 err = dispatcher.Register("test.comparer.compare",
                    ↪  compareFunc(differ, verifier, universityID, source))
109         }
```

```
110         if err == nil {
111                 err = dispatcher.Register("test.comparer.convert",
                    ↪  convertFunc(differ, verifier, universityID, source))
112         }
113         verifier.CheckSucceed(err)
114 }
```

Arquivo error.go

```
1  package testutil
2
3  import (
4          "bytes"
5          "io/ioutil"
6          "net/http"
7          "sync"
8  )
9
10 // Error returns a http.RoundTripper that returns the response of
   ↪  targetURL when
11 // requesting sourceURL
12 func Error(original http.RoundTripper, limit int, status int) Callbacker
   ↪  {
13         var lock sync.Mutex
14         var count int
15         return func(req *http.Request) (*http.Response, error) {
16                 lock.Lock()
17                 defer lock.Unlock()
18                 count++
19                 var response *http.Response
20                 var err error
21                 if count >= limit && err == nil {
22                         response = &http.Response{
23                                 Status:     http.StatusText(status),
24                                 StatusCode: status,
25                                 Body:
                                   ↪  ioutil.NopCloser(&bytes.Buffer{}),
26                                 Header:     http.Header{},
27                                 Request:    req,
28                                 Proto:      "HTTP/1.1",
```

```
29                                      ProtoMajor: 1,
30                                      ProtoMinor: 1,
31                                      Close:      true,
32                              }
33                      }
34                      if err == nil && response == nil {
35                              response, err = original.RoundTrip(req)
36                      }
37                      return response, err
38              }
39      }
```

Arquivo proxy.go

```
1   package testutil
2
3   import (
4           "net/http"
5           "net/url"
6           "sync"
7   )
8
9   // Proxy returns a http.RoundTripper that returns the response of
    ↪  targetURL when
10  // requesting sourceURL
11  func Proxy(original http.RoundTripper, limit int, sourceURL, targetURL
    ↪  string) Callbacker {
12          var lock sync.Mutex
13          var count int
14          return func(req *http.Request) (*http.Response, error) {
15                  lock.Lock()
16                  defer lock.Unlock()
17                  count++
18                  var response *http.Response
19                  var err error
20                  if count >= limit && err == nil && sourceURL ==
                    ↪  req.URL.String() {
21                          req.URL, err = url.Parse(targetURL)
22                  }
23                  if err == nil {
```

```
24                          response, err = original.RoundTrip(req)
25                  }
26                  return response, err
27          }
28  }
```

Arquivo recorder.go

```
1   package testutil
2
3   import (
4           "bufio"
5           "bytes"
6           "crypto/sha1"
7           "fmt"
8           "io"
9           "io/ioutil"
10          "net/http"
11          "net/http/httputil"
12          "net/url"
13          "sync"
14
15          "github.com/fjorgemota/gurudamatricula/util/coder"
16          "github.com/fjorgemota/gurudamatricula/util/kv"
17          "github.com/fjorgemota/gurudamatricula/util/pool"
18          "github.com/pkg/errors"
19  )
20
21  // RecorderMode defines the mode that the Recorder operates
22  type RecorderMode int8
23
24  const (
25          // ModeRecord is a mode where the requests/responses were saved
            ↪  on the
26          // database
27          ModeRecord RecorderMode = 2 >> iota
28          // ModeReplay is a mode where the requests are matched from the
            ↪  database
29          // and the responses are returned frmo it
30          ModeReplay
```

```
31          // ModeDisabled is a mode where the recorder do not do nothing
32          ModeDisabled
33  )
34
35  type recordRequest struct {
36          Body    []byte
37          Form    []byte
38          Headers []byte
39          URL     string
40          Method  string
41  }
42
43  // RecordRequest represents an http.Request read from the database
44  type RecordRequest struct {
45          Body    []byte
46          Form    url.Values
47          Headers http.Header
48          URL     string
49          Method  string
50  }
51
52  type recordResponse struct {
53          Body             []byte
54          Headers          []byte
55          Status           string
56          TransferEncoding []string
57          StatusCode       int
58          Proto            string
59          ProtoMajor       int
60          ProtoMinor       int
61  }
62
63  func toResponse(cod coder.Coder, req *http.Request, rec recordResponse)
    ↪  (*http.Response, error) {
64          var err error
65          var response *http.Response
66          var header http.Header
67          err = cod.DecodeFrom(bytes.NewReader(rec.Headers), &header)
68          if err == nil {
```

```
69                  response = &http.Response{
70                          Request:          req,
71                          Proto:            rec.Proto,
72                          ProtoMajor:       rec.ProtoMajor,
73                          ProtoMinor:       rec.ProtoMinor,
74                          Status:           rec.Status,
75                          StatusCode:       rec.StatusCode,
76                          Header:           header,
77                          TransferEncoding: rec.TransferEncoding,
78                          Close:            true,
79                          ContentLength:    int64(len(rec.Body)),
80                          Body:
                            ↪  ioutil.NopCloser(bytes.NewReader(rec.Body)),
81                  }
82          }
83          return response, err
84  }
85
86  type recordRequests struct {
87          Requests []string
88  }
89
90  type recorder struct {
91          orig          http.RoundTripper
92          mode          RecorderMode
93          store         kv.Store
94          lock          sync.RWMutex
95          cod           coder.Coder
96          hasher        func(req *http.Request) string
97          matcher       func(req *http.Request, i RecordRequest) bool
98          matchAll      bool
99          requestTable  string
100         responseTable string
101         metadataTable string
102 }
103
104 func (rec *recorder) SetMode(mode RecorderMode) {
105         rec.lock.Lock()
106         defer rec.lock.Unlock()
```

```
107         rec.mode = mode
108 }
109
110 func (rec *recorder) replay(req *http.Request, hash string)
    ↪  (*http.Response, error) {
111         var err error
112         var response recordResponse
113         var match bool
114         if rec.matchAll {
115                 err = rec.store.Get(rec.responseTable, hash, &response)
116                 match = true
117         } else {
118                 var record recordRequests
119                 err = rec.store.Get(rec.metadataTable, hash, &record)
120                 for _, reqKey := range record.Requests {
121                         var request RecordRequest
122                         if err == nil {
123                                 var reqBase recordRequest
124                                 err = rec.store.Get(rec.requestTable,
                                    ↪  reqKey, &reqBase)
125                                 if err == nil {
126                                         request.URL = reqBase.URL
127                                         request.Method = reqBase.Method
128                                         request.Body = reqBase.Body
129                                         err =
                                            ↪  rec.cod.DecodeFrom(bytes.NewReader(reqB
                                            ↪  &request.Form)
130                                 }
131                                 if err == nil {
132                                         err =
                                            ↪  rec.cod.DecodeFrom(bytes.NewReader(reqB
                                            ↪  &request.Headers)
133                                 }
134                         }
135                         match = false
136                         if err == nil {
137                                 match = rec.matcher(req, request)
138                         }
139                         if match {
```

```
140                                 err = rec.store.Get(rec.responseTable,
                                    ↪  reqKey, &response)
141                         }
142                         if err == nil && match {
143                                 break
144                         }
145                 }
146         }
147         if kv.IsKeyNotFoundError(err) {
148                 match = false
149                 err = nil
150         }
151         if match && err == nil {
152                 return toResponse(rec.cod, req, response)
153         }
154         return nil, err
155 }
156
157 func (rec *recorder) record(req *http.Request, hash string)
    ↪  (*http.Response, error) {
158         reqData, err := httputil.DumpRequestOut(req, true)
159         var reqCopy *http.Request
160         if err == nil {
161                 reqBuffer := bytes.NewBuffer(reqData)
162                 reqCopy, err =
                    ↪  http.ReadRequest(bufio.NewReader(reqBuffer))
163         }
164         var body bytes.Buffer
165         if err == nil {
166                 reqCopy.Body = ioutil.NopCloser(io.TeeReader(reqCopy.Body,
                    ↪  &body))
167         }
168         if err == nil {
169                 err = reqCopy.ParseForm()
170         }
171         if err == nil {
172                 _, err = io.Copy(ioutil.Discard, reqCopy.Body)
173         }
174         var resp *http.Response
```

```
175         if err == nil {
176                 resp, err = rec.orig.RoundTrip(req)
177         }
178         var bodyResp []byte
179         if err == nil {
180                 bodyResp, err = ioutil.ReadAll(resp.Body)
181         }
182         buf := pool.GetBytesBuffer()
183         defer pool.PutBytesBuffer(buf)
184         var reqHeaders, respHeaders, form int
185         if err == nil {
186                 err = rec.cod.EncodeTo(&req.Header, buf)
187                 reqHeaders = buf.Len()
188         }
189         if err == nil {
190                 err = rec.cod.EncodeTo(&req.Form, buf)
191                 form = buf.Len()
192         }
193         if err == nil {
194                 err = rec.cod.EncodeTo(&resp.Header, buf)
195                 respHeaders = buf.Len()
196         }
197         var recReq recordRequest
198         var recResp recordResponse
199         var reqKey string
200         bs := buf.Bytes()
201         if err == nil {
202                 recReq = recordRequest{
203                         URL:     req.URL.String(),
204                         Method:  req.Method,
205                         Headers: bs[:reqHeaders],
206                         Form:    bs[reqHeaders:form],
207                         Body:    body.Bytes(),
208                 }
209                 recResp = recordResponse{
210                         Proto:            resp.Proto,
211                         ProtoMinor:       resp.ProtoMinor,
212                         ProtoMajor:       resp.ProtoMajor,
213                         Status:           resp.Status,
```

```
214                     StatusCode:       resp.StatusCode,
215                     TransferEncoding: resp.TransferEncoding,
216                     Body:             bodyResp,
217                     Headers:          bs[form:respHeaders],
218             }
219             if !rec.matchAll {
220                     reqKey, err =
                        ↪ rec.store.GenerateID(rec.requestTable)
221             }
222     }
223     if err == nil {
224             if rec.matchAll {
225                     err = rec.store.Transaction([]kv.Reference{{Table:
                        ↪ rec.responseTable, Key: hash}}, func(db
                        ↪ kv.LimitedStore) error {
226                             var respCheck recordResponse
227                             errTransaction :=
                                ↪ db.Get(rec.responseTable, hash,
                                ↪ &respCheck)
228                             if kv.IsKeyNotFoundError(errTransaction)
                                ↪ {
229                                     // Only write to response table
                                        ↪ if it isn't already there
230                                     errTransaction =
                                        ↪ db.Set(rec.responseTable,
                                        ↪ hash, &recResp)
231                             }
232                             return errTransaction
233                     })
234             } else {
235                     err = rec.store.Transaction([]kv.Reference{{Table:
                        ↪ rec.requestTable, Key: reqKey}, {Table:
                        ↪ rec.responseTable, Key: reqKey}, {Table:
                        ↪ rec.metadataTable, Key: hash}}, func(db
                        ↪ kv.LimitedStore) error {
236                             errTransaction := db.Set(rec.requestTable,
                                ↪ reqKey, &recReq)
237                             if errTransaction == nil {
```

```
238                                     errTransaction =
                                        ↪ db.Set(rec.responseTable,
                                        ↪ reqKey, &recResp)
239                                 }
240                                 var requests recordRequests
241                                 if errTransaction == nil {
242                                     errTransaction =
                                        ↪ db.Get(rec.metadataTable,
                                        ↪ hash, &requests)
243                                 }
244                                 if kv.IsKeyNotFoundError(errTransaction)
                                    ↪ {
245                                     errTransaction = nil
246                                 }
247                                 if errTransaction == nil {
248                                     var count int
249                                     for _, requestKey := range
                                        ↪ requests.Requests {
250                                         if requestKey != reqKey {
251                                             count++
252                                         }
253                                     }
254                                     if len(requests.Requests) >=
                                        ↪ count {
255                                         requests.Requests =
                                            ↪ append(requests.Requests,
                                            ↪ reqKey)
256                                         errTransaction =
                                            ↪ db.Set(rec.metadataTable,
                                            ↪ hash, &requests)
257                                     }
258                                 }
259                                 return errTransaction
260                             })
261                     }
262             }
263             if err == nil {
264                     resp, err = toResponse(rec.cod, req, recResp)
265             }
```

```
266         return resp, err
267 }
268 func (rec *recorder) RoundTrip(req *http.Request) (*http.Response, error)
    ↪ {
269         if rec.mode == 0 {
270                 return rec.orig.RoundTrip(req)
271         }
272         rec.lock.RLock()
273         mode := rec.mode
274         rec.lock.RUnlock()
275         hash := rec.hasher(req)
276         var err error
277         var response *http.Response
278         if mode&ModeReplay > 0 {
279                 response, err = rec.replay(req, hash)
280         }
281         if mode&ModeRecord > 0 && response == nil {
282                 response, err = rec.record(req, hash)
283         }
284         if err == nil && response == nil {
285                 err = errors.New("testutil: Request not found on store")
286         }
287         if err != nil {
288                 response = nil
289         }
290         return response, err
291 }
292
293 // Recorder defines the public interface for an Recorder
294 type Recorder interface {
295         http.RoundTripper
296         SetMode(mode RecorderMode)
297 }
298
299 // RecorderOptions saves the options of the recorder
300 type RecorderOptions struct {
301         RequestsTable  string
302         ResponsesTable string
303         MetadataTable  string
```

```
304         Hasher          func(req *http.Request) string
305         Matcher         func(req *http.Request, i RecordRequest) bool
306         MatchAll        bool
307         Mode            RecorderMode
308 }
309
310 func getDefaultRecorderOptions(opts RecorderOptions) RecorderOptions {
311         if opts.Matcher == nil {
312                 opts.Matcher = func(req *http.Request, i RecordRequest)
                    ↪  bool {
313                         var body []byte
314                         var errReq error
315                         match := req.Method == i.Method &&
                            ↪  req.URL.String() == i.URL
316                         if match && req.Body != nil {
317                                 body, errReq = ioutil.ReadAll(req.Body)
318                                 if errReq == nil {
319                                         errReq = req.Body.Close()
320                                         req.Body =
                                            ↪  ioutil.NopCloser(bytes.NewReader(body))
321                                 }
322                         }
323                         return match && errReq == nil && bytes.Equal(body,
                            ↪  i.Body)
324                 }
325         }
326         if opts.Hasher == nil {
327                 opts.Hasher = func(req *http.Request) string {
328                         return fmt.Sprintf("%s-%x", req.Method,
                            ↪  sha1.Sum([]byte(req.URL.String())))
329                 }
330         }
331         if opts.RequestsTable == "" {
332                 opts.RequestsTable = "requests"
333         }
334         if opts.ResponsesTable == "" {
335                 opts.ResponsesTable = "responses"
336         }
337         if opts.MetadataTable == "" {
```

```
338                 opts.MetadataTable = "metadata"
339         }
340         return opts
341 }
342
343 // NewRecorder returns a new recorder
344 func NewRecorder(orig http.RoundTripper, store kv.Store, cod coder.Coder,
    ↪  opts RecorderOptions) Recorder {
345         opts = getDefaultRecorderOptions(opts)
346         return &recorder{
347                 orig:          orig,
348                 store:         store,
349                 cod:           cod,
350                 requestTable:  opts.RequestsTable,
351                 responseTable: opts.ResponsesTable,
352                 metadataTable: opts.MetadataTable,
353                 mode:          opts.Mode,
354                 hasher:        opts.Hasher,
355                 matcher:       opts.Matcher,
356                 matchAll:      opts.MatchAll,
357         }
358 }
```

Arquivo replacer.go

```
1  package testutil
2
3  import (
4          "bytes"
5          "io"
6          "io/ioutil"
7          "net/http"
8          "sync"
9
10         "github.com/fjorgemota/gurudamatricula/util/pool"
11 )
12
13 // Replacer returns a http.RoundTripper that replaces ocurrences from
    ↪  "source"
14 // to "target" after "limit" requests
```

```
15 func Replacer(original http.RoundTripper, limit int, source, target
   ↪  string) Callbacker {
16         var lock sync.Mutex
17         var count int
18         return func(req *http.Request) (*http.Response, error) {
19                 lock.Lock()
20                 defer lock.Unlock()
21                 count++
22                 response, err := original.RoundTrip(req)
23                 if count >= limit && err == nil {
24                         bs := pool.GetBytesBuffer()
25                         _, err = io.Copy(bs, response.Body)
26                         if err == nil {
27                                 err = response.Body.Close()
28                         }
29                         if err == nil {
30                                 s := bs.Bytes()
31                                 cs := bytes.Replace(s, []byte(source),
                                   ↪  []byte(target), -1)
32                                 bs.Reset()
33                                 _, err = bs.Write(cs)
34                                 response.Body = ioutil.NopCloser(bs)
35                         }
36                 }
37                 return response, err
38         }
39 }
```

Arquivo tmp.go

```
1  package testutil
2
3  import (
4          "bytes"
5          "compress/bzip2"
6          "compress/gzip"
7          "io"
8          "io/ioutil"
9          "os"
10         "path"
```

```
11          "strings"
12
13          "github.com/fjorgemota/gurudamatricula/util/coder"
14          "github.com/ulikunitz/xz"
15  )
16
17  type tmpDirOptions struct {
18          Path string
19          Base string
20  }
21  type tmpDir struct {
22          options tmpDirOptions
23  }
24
25  func (d tmpDir) copy(filename string) error {
26          filename = path.Join(d.options.Base, filename)
27          err := os.MkdirAll(path.Dir(filename), 0700)
28          var reader io.ReadCloser
29          var writer io.WriteCloser
30          var descompressor io.Reader
31          if err == nil {
32                  reader, err = os.Open(filename)
33          }
34          if err == nil {
35                  if strings.HasSuffix(filename, ".gz") {
36                          filename = strings.TrimSuffix(filename, ".gz")
37                          descompressor, err = gzip.NewReader(reader)
38                  } else if strings.HasSuffix(filename, ".bz2") {
39                          filename = strings.TrimSuffix(filename, ".bz2")
40                          descompressor = bzip2.NewReader(reader)
41                  } else if strings.HasSuffix(filename, ".xz") {
42                          filename = strings.TrimSuffix(filename, ".xz")
43                          descompressor, err = xz.NewReader(reader)
44                  } else {
45                          descompressor = reader
46                  }
47          }
48          if err == nil {
```

```
49                  writer, err = os.Create(path.Join(d.options.Path,
                    ↪  filename))
50          }
51          if err == nil {
52                  _, err = io.Copy(writer, descompressor)
53          }
54          if err == nil {
55                  err = reader.Close()
56          }
57          if err == nil {
58                  err = writer.Close()
59          }
60          return err
61  }
62
63  func (d tmpDir) GetPath(file string) string {
64          return path.Join(d.options.Path, d.options.Base, file)
65  }
66
67  func (d tmpDir) Open(file string) (*os.File, error) {
68          return os.Open(d.GetPath(file))
69  }
70
71  func (d tmpDir) Close() error {
72          var err error
73          if d.options.Path != "" {
74                  err = os.RemoveAll(d.options.Path)
75          }
76          return err
77  }
78
79  func (d tmpDir) EncodeTo(cod coder.Coder, target io.Writer) error {
80          return cod.EncodeTo(&d.options, target)
81  }
82
83  // TempDir defines an interface for managing temporary directories
84  type TempDir interface {
85          GetPath(file string) string
86          Open(file string) (*os.File, error)
```

```
87          Close() error
88  }
89
90  // LoadTempDir decodes options related to a temporary directory and
91  // verify if the directory exists
92  func LoadTempDir(data []byte, cod coder.Coder) (TempDir, error) {
93          var options tmpDirOptions
94          err := cod.DecodeFrom(bytes.NewReader(data), &options)
95          if err == nil {
96                  _, err = os.Stat(path.Join(options.Path, options.Base))
97          }
98          var result TempDir
99          if err == nil {
100                 result = tmpDir{options: options}
101         }
102         return result, err
103
104 }
105
106 // NewTempDir creates a temporary directory and descompress files to it
107 func NewTempDir(prefix, base string) (TempDir, error) {
108         tmpPath, err := ioutil.TempDir("", prefix)
109         var result tmpDir
110         if err == nil {
111                 result = tmpDir{
112                         options: tmpDirOptions{
113                                 Path: tmpPath,
114                                 Base: base,
115                         },
116                 }
117         }
118         if err == nil {
119                 err = os.Mkdir(path.Join(tmpPath, base), 0700)
120         }
121         if dir, exists := os.Stat(base); err == nil && exists == nil &&
        ↪  dir.IsDir() {
122                 var files []os.FileInfo
123                 files, err = ioutil.ReadDir(base)
124                 for _, file := range files {
```

```
125                         if err == nil {
126                                 err = result.copy(file.Name())
127                         }
128                 }
129         }
130         return result, err
131 }
```

B.1.34 Pasta robot/format/data

Arquivo schema.habilitation.json

```
1  {
2    "title": "Habilitation",
3    "description": "A habilitation of a course on a university",
4    "required": ["id", "name", "steps"],
5    "additionalProperties": false,
6    "type": "object",
7    "$schema": "http://json-schema.org/draft-07/schema",
8    "properties": {
9      "id": {
10       "type": "string"
11     },
12     "name": {
13       "type": "string"
14     },
15     "observation": {
16       "type": "string"
17     },
18     "course": {
19       "type": "object",
20       "additionalProperties": false,
21       "properties": {
22         "id": {
23           "type": "string"
24         },
25         "name": {
26           "type": "string"
27         },
28         "version": {
```

```
          "type": "string"
        },
        "url": {
          "type": "string",
          "format": "url"
        }
      }
    },
    "limits": {
      "type": "array",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["id", "name", "context", "unit", "values"],
        "properties": {
          "id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "context": {
            "type": "string",
            "enum": ["plan", "user"]
          },
          "unit": {
            "type": "string"
          },
          "values": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "min": {
                "type": "number",
                "minimum": 0
              },
              "ideal": {
                "type": "number",
                "minimum": 0
```

```
              },
              "max": {
                "type": "number",
                "minimum": 0
              }
            }
          }
        }
      },
      "tables": {
        "type": "array",
        "items": {
          "type": "object",
          "additionalProperties": false,
          "required": ["id", "title", "columns", "rows"],
          "properties": {
            "id": {
              "type": "string"
            },
            "title": {
              "type": "string"
            },
            "description": {
              "type": "string"
            },
            "columns": {
              "type": "array",
              "items": {
                "type": "object",
                "required": ["id"],
                "additionalProperties": false,
                "properties": {
                  "id": {
                    "type": "string"
                  },
                  "name": {
                    "type": "string"
                  }
```

```
            }
          }
        },
        "rows": {
          "type": "array",
          "items": {
            "type": "object",
            "additionalProperties": {
              "type": "string"
            },
            "properties": {}
          }
        }
      }
    },
    "steps": {
      "type": "array",
      "items": {
        "type": "object",
        "title": "Fase",
        "properties": {
          "id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "disciplines": {
            "type": "array",
            "items": {
              "type": "object",
              "required": ["id", "generic_id", "code", "name"],
              "properties": {
                "id": {
                  "type": "string"
                },
                "generic_id": {
                  "type": "string"
```

```
                },
                "code": {
                  "type": "string"
                },
                "name": {
                  "type": "string"
                },
                "description": {
                  "type": "string"
                },
                "type": {
                  "type": "string"
                },
                "workload": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "properties": {
                      "id": {
                        "type": "string"
                      },
                      "name": {
                        "type": "string"
                      },
                      "unit": {
                        "type": "string"
                      },
                      "value": {
                        "type": "number",
                        "minimum": 0
                      }
                    }
                  }
                },
                "tables": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "additionalProperties": false,
```

```
                "required": ["id", "title", "columns", "rows"],
                "properties": {
                  "id": {
                    "type": "string"
                  },
                  "title": {
                    "type": "string"
                  },
                  "description": {
                    "type": "string"
                  },
                  "columns": {
                    "type": "array",
                    "items": {
                      "type": "object",
                      "required": ["id"],
                      "additionalProperties": false,
                      "properties": {
                        "id": {
                          "type": "string"
                        },
                        "name": {
                          "type": "string"
                        }
                      }
                    }
                  },
                  "rows": {
                    "type": "array",
                    "items": {
                      "type": "object",
                      "additionalProperties": {
                        "type": "string"
                      },
                      "properties": {}
                    }
                  }
                }
```

```
                },
                "related": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "required": ["type", "items"],
                    "properties": {
                      "name": {
                        "type": "string"
                      },
                      "type": {
                        "type": "string",
                        "enum": ["prerequisite", "set", "equivalent"]
                      },
                      "items": {
                        "type": "array",
                        "items": {
                          "type": "object",
                          "required": ["type", "value"],
                          "additionalProperties": false,
                          "properties": {
                            "type": {
                              "type": "string",
                              "enum": [
                                "discipline_id",
                                "discipline_generic_id",
                                "text"
                              ]
                            },
                            "value": {
                              "type": "string"
                            },
                            "description": {
                              "type": "string"
                            }
                          }
                        }
                      }
                    }
```

```
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Arquivo schema.team.json

```
{
  "title": "Team",
  "description": "A team of a university",
  "required": ["id", "code", "discipline", "period", "campus"],
  "additionalProperties": false,
  "type": "object",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "properties": {
    "id": {
      "type": "string"
    },
    "code": {
      "type": "string"
    },
    "schedules": {
      "type": "array",
      "items": {
        "type": "object",
        "title": "Schedule",
        "required": ["start", "end", "dayOfWeek"],
        "additionalProperties": false,
        "properties": {
          "start": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "hour": {
```

```
                "type": "integer",
                "maximum": 23,
                "minimum": 0
              },
              "minute": {
                "type": "integer",
                "maximum": 59,
                "minimum": 0
              }
            }
          },
          "end": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "hour": {
                "type": "integer",
                "maximum": 23,
                "minimum": 0
              },
              "minute": {
                "type": "integer",
                "maximum": 59,
                "minimum": 0
              }
            }
          },
          "dayOfWeek": {
            "type": "integer",
            "maximum": 7,
            "minimum": 1
          },
          "room": {
            "type": "object",
            "required": ["id", "name"],
            "additionalProperties": false,
            "properties": {
              "id": {
                "type": "string"
```

```
              },
              "name": {
                "type": "string"
              },
              "osm_id": {
                "type": "integer"
              },
              "description": {
                "type": "string"
              }
            }
          }
        }
      }
    },
    "teachers": {
      "type": "array",
      "items": {
        "type": "object",
        "title": "Teacher",
        "required": ["id", "name"],
        "additionalProperties": false,
        "properties": {
          "id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "url": {
            "type": "string",
            "format": "uri"
          }
        }
      }
    },
    "campus": {
      "type": "object",
      "additionalProperties": false,
```

```
      "required": ["id", "name"],
      "properties": {
        "id": {
          "type": "string"
        },
        "name": {
          "type": "string"
        },
        "latitude": {
          "type": "number",
          "maximum": 90,
          "minimum": -90
        },
        "longitude": {
          "type": "number",
          "maximum": 180,
          "minimum": -180
        }
      }
    },
    "discipline": {
      "type": "object",
      "additionalProperties": false,
      "required": ["id", "code", "name"],
      "properties": {
        "id": {
          "type": "string"
        },
        "generic_id": {
          "type": "string"
        },
        "code": {
          "type": "string"
        },
        "name": {
          "type": "string"
        }
      }
    },
```

```
    "period": {
      "type": "object",
      "additionalProperties": false,
      "required": ["id", "name"],
      "properties": {
        "id": {
          "type": "string"
        },
        "name": {
          "type": "string"
        }
      }
    },
    "vacancies": {
      "type": "object",
      "additionalProperties": false,
      "required": ["filled", "offered"],
      "properties": {
        "filled": {
          "type": "integer"
        },
        "offered": {
          "type": "integer"
        }
      }
    },
    "tables": {
      "type": "array",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["id", "title", "columns", "rows"],
        "properties": {
          "id": {
            "type": "string"
          },
          "title": {
            "type": "string"
          },
```

```
184           "description": {
185             "type": "string"
186           },
187           "columns": {
188             "type": "array",
189             "items": {
190               "type": "object",
191               "required": ["id"],
192               "additionalProperties": false,
193               "properties": {
194                 "id": {
195                   "type": "string"
196                 },
197                 "name": {
198                   "type": "string"
199                 }
200               }
201             }
202           },
203           "rows": {
204             "type": "array",
205             "items": {
206               "type": "object",
207               "additionalProperties": {
208                 "type": "string"
209               },
210               "properties": {}
211             }
212           }
213         }
214       }
215     },
216     "course": {
217       "type": "object",
218       "additionalProperties": false,
219       "properties": {
220         "id": {
221           "type": "string"
222         },
```

```
        "name": {
          "type": "string"
        },
        "exclusive": {
          "type": "string",
          "default": "unknown",
          "enum": ["yes", "no", "unknown"]
        }
      }
    }
  }
}
```

B.1.35 Pasta robot/format/pool

Arquivo habilitation.go

```
package pool

import (
        "sync"

        "github.com/fjorgemota/gurudamatricula/robot/format"
)

var habilitationPool = sync.Pool{
        New: func() interface{} {
                return new(format.Habilitation)
        },
}

func GetHabilitation() *format.Habilitation {
        return habilitationPool.Get().(*format.Habilitation)
}

func CopyHabilitation(target, source *format.Habilitation) {
        limits := target.Limits
        steps := target.Steps
        tables := target.Tables

```

```
24          *target = *source
25          limits = limits[:cap(limits)]
26          if len(limits) < len(source.Limits) {
27                  limits = append(limits, make([]format.Limit,
                    ↪  len(source.Limits)-len(limits))...)
28          }
29          copy(limits, source.Limits)
30          target.Limits = limits[:len(source.Limits)]
31
32          steps = steps[:cap(steps)]
33          if len(steps) < len(source.Steps) {
34                  steps = append(steps, make([]format.Step,
                    ↪  len(source.Steps)-len(steps))...)
35          }
36          for i := range source.Steps {
37                  disciplines := steps[i].Disciplines
38                  sourceDisciplines := source.Steps[i].Disciplines
39                  disciplines = disciplines[:cap(disciplines)]
40                  if len(disciplines) < len(sourceDisciplines) {
41                          disciplines = append(disciplines,
                            ↪  make([]format.Discipline,
                            ↪  len(sourceDisciplines)-len(disciplines))...)
42                  }
43                  for i := range sourceDisciplines {
44                          CopyDiscipline(&disciplines[i],
                            ↪  &sourceDisciplines[i])
45                  }
46                  steps[i] = source.Steps[i]
47                  steps[i].Disciplines =
                    ↪  disciplines[:len(sourceDisciplines)]
48          }
49          target.Steps = steps[:len(source.Steps)]
50
51          tables = tables[:cap(tables)]
52          if len(tables) < len(source.Tables) {
53                  tables = append(tables, make([]format.Table,
                    ↪  len(source.Tables)-len(tables))...)
54          }
55          for j := range source.Tables {
```

```
56                 CopyTable(&tables[j], &source.Tables[j])
57         }
58         target.Tables = tables[:len(source.Tables)]
59 }
60
61 func CopyDiscipline(target, source *format.Discipline) {
62         related := target.Related
63         workload := target.Workload
64         tables := target.Tables
65
66         *target = *source
67         workload = workload[:cap(workload)]
68         if len(workload) < len(source.Workload) {
69                 workload = append(workload,
                   ↪  make([]format.DisciplineWorkload,
                   ↪  len(source.Workload)-len(workload))...)
70         }
71         copy(workload, source.Workload)
72         target.Workload = workload[:len(source.Workload)]
73
74         related = related[:cap(related)]
75         if len(related) < len(source.Related) {
76                 related = append(related, make([]format.DisciplineRelated,
                   ↪  len(source.Related)-len(related))...)
77         }
78         for i := range source.Related {
79                 items := related[i].Items
80                 sourceItems := source.Related[i].Items
81                 items = items[:cap(items)]
82                 if len(items) < len(sourceItems) {
83                         items = append(items,
                           ↪  make([]format.DisciplineRelatedItem,
                           ↪  len(sourceItems)-len(items))...)
84                 }
85                 copy(items, sourceItems)
86                 related[i] = source.Related[i]
87                 related[i].Items = items[:len(sourceItems)]
88         }
89         target.Related = related[:len(source.Related)]
```

```
90
91          tables = tables[:cap(tables)]
92          if len(tables) < len(source.Tables) {
93                  tables = append(tables, make([]format.Table,
                    ↪  len(source.Tables)-len(tables))...)
94          }
95          for i := range source.Tables {
96                  CopyTable(&tables[i], &source.Tables[i])
97          }
98          target.Tables = tables[:len(source.Tables)]
99  }
100
101 func CleanDiscipline(discipline *format.Discipline) {
102         tables := discipline.Tables
103         related := discipline.Related
104         workload := discipline.Workload
105         for i := range tables {
106                 CleanTable(&tables[i])
107         }
108         for i := range related {
109                 items := related[i].Items
110                 for j := range items {
111                         items[j] = format.DisciplineRelatedItem{}
112                 }
113                 related[i] = format.DisciplineRelated{
114                         Items: items,
115                 }
116         }
117         for i := range workload {
118                 workload[i] = format.DisciplineWorkload{}
119         }
120         *discipline = format.Discipline{
121                 Workload: workload[:0],
122                 Related:  related[:0],
123                 Tables:   tables[:0],
124         }
125 }
126
127 func CleanHabilitation(habilitation *format.Habilitation) {
```

```
128         limits := habilitation.Limits
129         steps := habilitation.Steps
130         tables := habilitation.Tables
131         for i := range tables {
132                 CleanTable(&tables[i])
133         }
134         for i := range limits {
135                 limits[i] = format.Limit{}
136         }
137         for i := range steps {
138                 disciplines := steps[i].Disciplines
139                 for j := range disciplines {
140                         CleanDiscipline(&disciplines[j])
141                 }
142                 steps[i] = format.Step{
143                         Disciplines: disciplines[:0],
144                 }
145         }
146         *habilitation = format.Habilitation{
147                 Limits: limits[:0],
148                 Steps:  steps[:0],
149                 Tables: tables[:0],
150         }
151 }
152
153 func PutHabilitation(habilitation *format.Habilitation) {
154         CleanHabilitation(habilitation)
155         habilitationPool.Put(habilitation)
156 }
```

Arquivo table.go

```
1  package pool
2
3  import "github.com/fjorgemota/gurudamatricula/robot/format"
4
5  func CleanTable(table *format.Table) {
6          columns := table.Columns
7          for j := range columns {
8                  columns[j] = format.Column{}
```

```
9           }
10          rows := table.Rows
11          for j := range rows {
12                  for key := range rows[j] {
13                          delete(rows[j], key)
14                  }
15          }
16          *table = format.Table{
17                  Rows:    rows[:0],
18                  Columns: columns[:0],
19          }
20  }
21
22  func CopyTable(target, source *format.Table) {
23          columns := target.Columns
24          rows := target.Rows
25          *target = *source
26
27          columns = columns[:cap(columns)]
28          if len(columns) < len(source.Columns) {
29                  columns = append(columns, make([]format.Column,
                    ↪  len(source.Columns)-len(columns))...)
30          }
31          copy(columns, source.Columns)
32          target.Columns = columns[:len(source.Columns)]
33          rows = rows[:cap(rows)]
34          if len(rows) < len(source.Rows) {
35                  rows = append(rows, make([]map[string]string,
                    ↪  len(source.Rows)-len(rows))...)
36          }
37          for j := range source.Rows {
38                  if rows[j] == nil {
39                          rows[j] = make(map[string]string,
                            ↪  len(source.Rows[j]))
40                  }
41                  for key := range source.Rows[j] {
42                          rows[j][key] = source.Rows[j][key]
43                  }
44          }
```

```
45          target.Rows = rows[:len(source.Rows)]
46  }
```

Arquivo team.go

```
1   package pool
2
3   import (
4           "sync"
5
6           "github.com/fjorgemota/gurudamatricula/robot/format"
7   )
8
9   var teamPool = sync.Pool{
10          New: func() interface{} {
11                  return new(format.Team)
12          },
13  }
14
15  func GetTeam() *format.Team {
16          return teamPool.Get().(*format.Team)
17  }
18
19  func CopyTeam(target, source *format.Team) {
20          schedules := target.Schedules
21          teachers := target.Teachers
22          tables := target.Tables
23          *target = *source
24          schedules = schedules[:cap(schedules)]
25          if len(schedules) < len(source.Schedules) {
26                  schedules = append(schedules, make([]format.Schedule,
                    ↪  len(source.Schedules)-len(schedules))...)
27          }
28          copy(schedules, source.Schedules)
29          target.Schedules = schedules[:len(source.Schedules)]
30
31          teachers = teachers[:cap(teachers)]
32          if len(teachers) < len(source.Teachers) {
33                  teachers = append(teachers, make([]format.Teacher,
                    ↪  len(source.Teachers)-len(teachers))...)
```

```
34          }
35          copy(teachers, source.Teachers)
36          target.Teachers = teachers[:len(source.Teachers)]
37
38          tables = tables[:cap(tables)]
39          if len(tables) < len(source.Tables) {
40                  tables = append(tables, make([]format.Table,
                    ↪  len(source.Tables)-len(tables))...)
41          }
42          for j := range source.Tables {
43                  CopyTable(&tables[j], &source.Tables[j])
44          }
45          target.Tables = tables[:len(source.Tables)]
46  }
47
48  func CleanTeam(team *format.Team) {
49          schedules := team.Schedules
50          teachers := team.Teachers
51          tables := team.Tables
52          for i := range schedules {
53                  schedules[i] = format.Schedule{}
54          }
55          for i := range teachers {
56                  teachers[i] = format.Teacher{}
57          }
58          for i := range tables {
59                  CleanTable(&tables[i])
60          }
61          *team = format.Team{
62                  Schedules: schedules[:0],
63                  Teachers:  teachers[:0],
64                  Tables:    tables[:0],
65          }
66  }
67
68  func PutTeam(team *format.Team) {
69          CleanTeam(team)
70          teamPool.Put(team)
71  }
```

B.1.36 Pasta robot/util/attrs

Arquivo attrs.go

```
1  package attrs
2
3  import (
4          "bytes"
5
6          "github.com/pkg/errors"
7          "golang.org/x/net/html"
8          "golang.org/x/net/html/atom"
9  )
10
11 func GetAttrs(doc *html.Tokenizer, selected map[string]struct{}, values
   ↪  map[string]string, hasNextAttr bool) (map[string]string, error) {
12         var key, value []byte
13         for hasNextAttr == true {
14                 key, value, hasNextAttr = doc.TagAttr()
15                 if _, ok := selected[string(key)]; ok {
16                         if _, ok = values[string(key)]; ok {
17                                 delete(values, string(key))
18                                 return values,
                                   ↪  errors.WithStack(errMultipleAttr)
19                         }
20                         values[string(key)] = string(value)
21                 }
22         }
23         return values, nil
24 }
25
26 func GetAttrBytes(doc *html.Tokenizer, attr string, hasNextAttr bool, buf
   ↪  *bytes.Buffer) ([]byte, error) {
27         var key, value []byte
28         buf.Reset()
29         for hasNextAttr == true {
30                 key, value, hasNextAttr = doc.TagAttr()
31                 if atom.String(key) == attr {
32                         if buf.Len() > 0 {
```

```
33                                  return nil,
                                    ↪  errors.WithStack(errMultipleAttr)
34                          }
35                          buf.Write(value)
36                  }
37          }
38          return buf.Bytes(), nil
39  }
```

Arquivo base.go

```
1  package attrs
2
3  import "errors"
4
5  var errMultipleAttr = errors.New("attrs: Found multiple attributes with
   ↪  same name")
```

Arquivo errors.go

```
1  package attrs
2
3  import "github.com/pkg/errors"
4
5  func IsMultipleAttrError(err error) bool {
6          return errors.Cause(err) == errMultipleAttr
7  }
```

B.1.37 Pasta robot/util/ccookiejar

Arquivo cookie.go

```
1  package ccookiejar
2
3  import (
4          "bytes"
5          "io"
6          "net/http"
7          "net/http/cookiejar"
8          "net/url"
```

```
9           "sync"
10
11          "github.com/fjorgemota/gurudamatricula/util/coder"
12  )
13
14  // CodedCookie represents a CookieJar that can encode and decode itself
15  // automatically
16  type CodedCookieJar interface {
17          coder.Decoder
18          coder.Encoder
19          http.CookieJar
20  }
21
22  var zero struct{}
23
24  type cookieJar struct {
25          jar  http.CookieJar
26          urls map[string]struct{}
27          lock sync.Mutex
28  }
29
30  func (cj *cookieJar) SetCookies(u *url.URL, cookies []*http.Cookie) {
31          cj.lock.Lock()
32          defer cj.lock.Unlock()
33          cj.jar.SetCookies(u, cookies)
34          if cookies := cj.Cookies(u); len(cookies) > 0 {
35                  cj.urls[u.String()] = zero
36          }
37  }
38
39  func (cj *cookieJar) Cookies(u *url.URL) []*http.Cookie {
40          return cj.jar.Cookies(u)
41  }
42
43  func (cj *cookieJar) EncodeTo(cod coder.Coder, writer io.Writer) error {
44          cj.lock.Lock()
45          defer cj.lock.Unlock()
46          results := make(map[string][]byte, len(cj.urls))
47          cod.Register(results)
```

```
48          var err error
49          for urlString := range cj.urls {
50                  var urlInstance *url.URL
51                  if err == nil {
52                          urlInstance, err = url.Parse(urlString)
53                  }
54                  if err == nil {
55                          cookies := cj.Cookies(urlInstance)
56                          if len(cookies) > 0 {
57                                  var buf bytes.Buffer
58                                  err = cod.EncodeTo(cookies, &buf)
59                                  if err == nil {
60                                          results[urlString] = buf.Bytes()
61                                  }
62                          }
63                  }
64          }
65          if err == nil {
66                  err = cod.EncodeTo(results, writer)
67          }
68          return err
69  }
70
71  func (cj *cookieJar) DecodeFrom(cod coder.Coder, reader io.Reader) error
    ↪ {
72          cj.lock.Lock()
73          defer cj.lock.Unlock()
74          results := make(map[string][]byte)
75          err := cod.DecodeFrom(reader, &results)
76          for urlString, data := range results {
77                  var cookies []*http.Cookie
78                  if err == nil {
79                          err = cod.DecodeFrom(bytes.NewReader(data),
                            ↪  &cookies)
80                  }
81                  var urlInstance *url.URL
82                  if err == nil {
83                          urlInstance, err = url.Parse(urlString)
84                  }
```

```
85                  if err == nil {
86                          cj.jar.SetCookies(urlInstance, cookies)
87                          cj.urls[urlString] = zero
88                  }
89          }
90          return err
91  }
92
93  // NewCookieJar returns a new cookiejar that can be encoded and decoded
94  // by the coder package
95  func NewCookieJar(jar http.CookieJar) CodedCookieJar {
96          return &cookieJar{jar: jar, urls: make(map[string]struct{})}
97  }
98
99  // NewCookieJarWithOptions returns a new cookiejjar that can be encoded
    ↪  and
100 // decoded by the coder package but based on a empty cookiejar with the
101 // passed options
102 func NewCookieJarWithOptions(options *cookiejar.Options) (CodedCookieJar,
    ↪  error) {
103         var jar CodedCookieJar
104         var err error
105         var underlingCookieJar http.CookieJar
106         underlingCookieJar, err = cookiejar.New(options)
107         if err == nil {
108                 jar = NewCookieJar(underlingCookieJar)
109         }
110         return jar, err
111 }
```

B.1.38 Pasta robot/util/uspreq

Arquivo base.go

```
1  package uspreq
2
3  import "errors"
4
5  const campiIndex =
    ↪  "https://uspdigital.usp.br/jupiterweb/jupColegiadoLista?tipo=D"
```

```
6
7  // ErrInvalidStatus is returned when the HTTP Status returned by the
   ↪  server is
8  // not on the range between 200 and 299 (including these numbers)
9  var ErrInvalidStatus = errors.New("usp2: Invalid HTTP Status detected")
```

Arquivo campi.go

```
1  package uspreq
2
3  import (
4          "bytes"
5          "io"
6          "net/http"
7          "net/url"
8
9          "github.com/fjorgemota/gurudamatricula/robot/format"
10         "github.com/fjorgemota/gurudamatricula/util/pool"
11         "golang.org/x/net/html"
12         "golang.org/x/net/html/atom"
13 )
14
15 // GetCampi returns a slice of format.Campus with the campus found on USP
16 // university
17 func GetCampi(client *http.Client, ID, userAgent string) ([]format.Campus,
   ↪  error) {
18         req, err := GetRequest(campiIndex, url.Values{
19                 "tipo": []string{"D"},
20         }, userAgent)
21         var resp *http.Response
22         if err == nil {
23                 resp, err = client.Do(req)
24         }
25         var reader io.Reader
26         if err == nil {
27                 reader, err = GetDoc(req, resp, ID)
28         }
29         var doc *html.Tokenizer
30         if err == nil {
31                 doc = html.NewTokenizer(reader)
```

```
32          }
33          var campi []format.Campus
34          if err == nil {
35                  columnIndex := -1
36                  columnCount := -1
37                  tmp := pool.GetBytesBuffer()
38                  var campus format.Campus
39                  for err == nil {
40                          tagType := doc.Next()
41                          if tagType == html.ErrorToken {
42                                  err = doc.Err()
43                          } else if tagType == html.StartTagToken {
44                                  tag, _ := doc.TagName()
45                                  tagName := atom.String(tag)
46                                  switch tagName {
47                                  case "table":
48                                          columnIndex = 0
49                                  case "tr":
50                                          columnIndex = 0
51                                  case "td":
52                                          columnIndex++
53                                  }
54                          } else if tagType == html.EndTagToken {
55                                  tag, _ := doc.TagName()
56                                  tagName := atom.String(tag)
57                                  switch tagName {
58                                  case "table":
59                                          columnIndex = -1
60                                          columnCount = -1
61                                  case "tr":
62                                          if columnCount == 2 {
63                                                  if campus.ID != "" &&
                                                    ↪  campus.Name != "" {
64                                                          campi =
                                                            ↪  append(campi,
                                                            ↪  campus)
65                                                  }
66                                                  campus = format.Campus{}
67                                          }
```

```
68                                      columnCount = columnIndex
69                              case "td":
70                                      if columnCount != 2 ||
                                        ↪  columnIndex >= 3 || tmp.Len()
                                        ↪  == 0 {
71                                              continue
72                                      }
73                                      switch columnIndex {
74                                      case 1:
75                                              campus.ID = tmp.String()
76                                      case 2:
77                                              campus.Name =
                                                ↪  tmp.String()
78                                      }
79                                      tmp.Reset()
80                              }
81                      } else if tagType == html.TextToken &&
                        ↪  columnCount == 2 {
82                              tmp.Write(bytes.TrimSpace(doc.Text()))
83                      }
84              }
85              pool.PutBytesBuffer(tmp)
86      }
87      if err == io.EOF {
88              err = nil
89      }
90      return campi, err
91  }
```

Arquivo reader.go

```
1  package uspreq
2
3  import (
4          "io"
5  )
6
7  type fetcherReader struct {
8          response io.ReadCloser
9          reader   io.Reader
```

```
}

func (fr *fetcherReader) Read(b []byte) (int, error) {
        n, err := fr.reader.Read(b)
        if err == io.EOF {
                err = fr.response.Close()
                fr.reader = nil
                if err == nil {
                        // If there is no errors, just return EOF
                        err = io.EOF
                }
        }
        return n, err
}
```

Arquivo requests.go

```
package uspreq

import (
        "io"
        "net/http"
        "net/url"

        "golang.org/x/text/encoding/charmap"
)

// GetRequest prepares a http.Request for the USP university
func GetRequest(url string, params url.Values, userAgent string)
↪  (*http.Request, error) {
        req, err := http.NewRequest("GET", url, nil)
        if err == nil {
                params.Set("print", "true")
                req.URL.RawQuery = params.Encode()
                req.Header.Add("User-Agent", userAgent)
                req.Header.Add("Pragma", "no-cache")
                req.Header.Add("Cache-Control", "no-cache")
        }
        return req, err
}
```

```
23
24 // GetDoc checks if the status code is valid, and, if so, prepares the
   ↪ body
25 // of the response to be read correctly and checked against errors,
   ↪ returning
26 // a FetcherReader after that preparation
27 func GetDoc(req *http.Request, response *http.Response, ID string)
   ↪ (io.Reader, error) {
28         err := ErrInvalidStatus
29         var fr io.Reader
30         if response.StatusCode >= 200 && response.StatusCode <= 299 {
31                 resp := response.Body
32                 // Decode the content of the page from ISO-8859-1 to
                   ↪ UTF-8
33                 decoder := charmap.ISO8859_1.NewDecoder()
34                 reader := decoder.Reader(resp)
35                 fr = &fetcherReader{
36                         response: resp,
37                         reader:   reader,
38                 }
39                 err = nil
40         }
41         return fr, err
42 }
```

Arquivo start.go

```
1  package uspreq
2
3  import (
4          "net/http"
5
6          uuid "github.com/gofrs/uuid"
7
8          "github.com/fjorgemota/gurudamatricula/robot/format"
9          "github.com/sirupsen/logrus"
10 )
11
```

```
12 func Start(logger logrus.FieldLogger, client *http.Client, dispatch
   ↪  func(campi []format.Campus, ID string) error, parallelRequests int,
   ↪  userAgent string) error {
13         handle, err := uuid.NewV4()
14         var ID string
15         if err == nil {
16                 ID = handle.String()
17         }
18         var campi []format.Campus
19         if err == nil {
20                 campi, err = GetCampi(client, ID, userAgent)
21         }
22         splitPoint := len(campi) / parallelRequests
23         logger.WithField("number_of_campi",
           ↪  len(campi)).WithField("parallel_requests",
           ↪  parallelRequests).WithField("split_point",
           ↪  splitPoint).Debug("Dispatching tasks to get disciplines")
24         batch := make([]format.Campus, 0, splitPoint)
25         for _, campus := range campi {
26                 if len(batch) > splitPoint {
27                         err = dispatch(batch, ID)
28                         batch = batch[:0]
29                 }
30                 if err == nil {
31                         batch = append(batch, campus)
32                 }
33         }
34         if len(batch) > 0 && err == nil {
35                 err = dispatch(batch, ID)
36         }
37         return err
38 }
```

B.1.39 Pasta util/indexer/stringset

Arquivo stringset.go

```
1 package stringset
2
3 // This code is based on code from https://github.com/smartystreets/mafsa
```

```
4  // But optimized with new Go utilities from the standard library and
5  // with a few more public variables so we can access the tree correctly
6  import (
7          "bytes"
8          "errors"
9          "sort"
10         "strconv"
11         "sync"
12
13         "github.com/fjorgemota/gurudamatricula/util/pool"
14 )
15
16 type (
17         // Builder implements an MA-FSA that contains all the
           ↪  "scaffolding" (state)
18         // necessary to perform optimizations after inserting each item,
           ↪  which prevents
19         // the tree from growing too big.
20         Builder struct {
21                 Root      *Node
22                 idCounter int
23                 register  map[string]*Node
24                 tmpList   []int
25                 tmpBytes  []byte
26                 buf       *bytes.Buffer
27         }
28
29         // Node is a node in a Builder that contains all the
30         // info necessary to be a part of optimizations during item
           ↪  insertions.
31         Node struct {
32                 edges        map[rune]*Node
33                 id           int
34                 char         rune
35                 lastChildKey rune
36                 final        bool
37         }
38 )
39
```

```
40 var nodePool = sync.Pool{
41         New: func() interface{} {
42                 return &Node{
43                         edges: make(map[rune]*Node),
44                 }
45         },
46 }
47
48 func getNode(char rune, id int) *Node {
49         result := nodePool.Get().(*Node)
50         result.char = char
51         result.id = id
52         result.final = false
53         result.lastChildKey = 0
54         return result
55 }
56
57 func putNode(node *Node) {
58         for key := range node.edges {
59                 delete(node.edges, key)
60         }
61         nodePool.Put(node)
62 }
63
64 // Insert adds val to the tree and performs optimizations to minimize
65 // the number of nodes in the tree. The inserted val must be
66 // lexicographically equal to or higher than the last inserted val.
67 func (t *Builder) InsertPrefix(lastState *Node, value string) (*Node,
    ↪  error) {
68         // Establish prefix shared between this and the last word
69         // and traverse the tree up to the differing part (suffix)
70         commonPrefixLen := -1
71         if lastState == nil {
72                 lastState = t.Root
73         }
74         word := []rune(value)
75
76         for index, char := range word {
77                 if lastState.lastChildKey != char {
```

```
78                          if lastState.lastChildKey > char {
79                                  return nil, errors.New("Insertions must
                                    ↪  be performed in lexicographical
                                    ↪  order")
80                          }
81                          commonPrefixLen = index
82                          break
83                  }
84                  lastState = lastState.edges[char]
85          }
86
87          if commonPrefixLen < 0 {
88                  return lastState, nil
89          }
90
91          // Perform optimization steps
92          err := t.replaceOrRegister(lastState)
93
94          // Add the differing part (suffix) to the tree
95          for _, char := range word[commonPrefixLen:] {
96                  newNode := getNode(char, t.idCounter)
97                  lastState.edges[char] = newNode
98                  lastState.lastChildKey = char
99                  lastState = newNode
100                 t.idCounter++
101         }
102         return lastState, err
103 }
104
105 func (t *Builder) InsertWord(lastState *Node, value string) (*Node,
    ↪  error) {
106         node, err := t.InsertPrefix(lastState, value)
107         if err == nil {
108                 node.final = true
109         }
110         return node, err
111 }
112
113 // Finish completes the optimizations on a tree. You must call Finish
```

```
114 // at least once, like immediately after all entries have been inserted.
115 func (t *Builder) Finish() error {
116         err := t.replaceOrRegister(t.Root)
117         pool.PutBytesBuffer(t.buf)
118         t.buf = nil
119         return err
120 }
121
122 // replaceOrRegister minimizes the number of nodes in the tree
123 // starting with leaf nodes below state.
124 func (t *Builder) replaceOrRegister(state *Node) error {
125         var err error
126         if len(state.edges) == 0 {
127                 return err
128         }
129
130         child := state.edges[state.lastChildKey]
131
132         if err == nil {
133                 err = t.replaceOrRegister(child)
134         }
135         // If there exists a state q in the tree such that
136         // it is in the register and equivalent
137         // to (duplicate of) the child:
138         //         1) Set the state's lastChildKey to q
139         //        2) add the child to the pool again
140         // Otherwise, add child to the register.
141         if err == nil {
142                 err = child.hash(t)
143         }
144         if err == nil {
145                 bs := t.buf.Bytes()
146                 if equiv, ok := t.register[string(bs)]; ok {
147                         state.edges[equiv.char] = equiv
148                         putNode(child)
149                 } else {
150                         t.register[string(bs)] = child
151                 }
152         }
```

```
153         t.buf.Reset()
154         return err
155 }
156
157 // GetNodes return all the nodes found on the tree, using an optimized
158 // algorithm
159 func (t *Builder) GetNodes() map[*Node]int {
160         mapping := make(map[*Node]int, t.idCounter)
161         oldCount := len(mapping)
162         mapping[t.Root] = oldCount
163         newCount := len(mapping)
164         for oldCount < newCount {
165                 oldCount = newCount
166                 for node := range mapping {
167                         for _, edge := range node.edges {
168                                 if _, ok := mapping[edge]; !ok {
169                                         mapping[edge] = len(mapping)
170                                 }
171                         }
172                 }
173                 newCount = len(mapping)
174         }
175         return mapping
176 }
177
178 func (tn *Node) Final() bool {
179         return tn.final
180 }
181
182 func (tn *Node) Char() rune {
183         return tn.char
184 }
185
186 func (tn *Node) Edges() map[rune]*Node {
187         return tn.edges
188 }
189
190 // hash returns a string representation of this node's equivalence class,
191 // not the unique node itself. A node is equivalent to another node if:
```

```
192 //  - Their incoming edge is the same (their char value is equal)
193 //  - They are both final or both nonfinal
194 //  - They have the same number of outgoing edges
195 //  - Their outgoing edges go to the same nodes, respectively
196 func (tn *Node) hash(t *Builder) error {
197         _, err := t.buf.WriteRune(tn.char)
198         if err == nil {
199                 err = t.buf.WriteByte('|')
200         }
201         if err == nil {
202                 if tn.final {
203                         err = t.buf.WriteByte('1')
204                 } else {
205                         err = t.buf.WriteByte('0')
206                 }
207         }
208 
209         // Iterating a map is not deterministic in its ordering,
210         // so we have to copy the values into a slice and sort it.
211         for _, child := range tn.edges {
212                 t.tmpList = append(t.tmpList, child.id)
213         }
214         sort.Ints(t.tmpList)
215         for _, id := range t.tmpList {
216                 if err == nil {
217                         err = t.buf.WriteByte('|')
218                 }
219                 if err == nil {
220                         t.tmpBytes = strconv.AppendInt(t.tmpBytes[:0],
                        ↪  int64(id), 10)
221                         _, err = t.buf.Write(t.tmpBytes)
222                 }
223         }
224         t.tmpList = t.tmpList[:0]
225         return err
226 }
227 
228 // New constructs a new, empty MA-FSA that can be filled with data.
229 func New() *Builder {
```

```
230         t := new(Builder)
231         t.register = make(map[string]*Node)
232         t.Root = getNode(0, t.idCounter)
233         t.buf = pool.GetBytesBuffer()
234         t.idCounter++
235         return t
236 }
```

B.1.40 Pasta util/testutil/aeinstancetest

Arquivo base.go

```
1  package aeinstancetest
2
3  import "google.golang.org/appengine/aetest"
4
5  type Handler interface {
6          Skip(err error)
7          Check(err error)
8  }
9
10 type Manager interface {
11         CloseInstance()
12         GetInstance(opts *aetest.Options) aetest.Instance
13 }
```

Arquivo ginkgo.go

```
1  package aeinstancetest
2
3  import (
4          "fmt"
5
6          . "github.com/onsi/ginkgo"
7          . "github.com/onsi/gomega"
8  )
9
10 type ginkgoHandler struct{}
11
12 func (gh ginkgoHandler) Skip(err error) {
```

```
13          // This causes some tests to skip because of ginkgo...but okay.
14          Skip(
15                  fmt.Sprintf(
16                          "There's no way to start the AppEngine's instance
                            ↪  now: %s",
17                          err,
18                  ),
19          )
20  }
21
22  func (gh ginkgoHandler) Check(err error) {
23          Expect(err).To(Succeed())
24  }
25
26  func NewGinkgoHandler() Handler {
27          return ginkgoHandler{}
28  }
```

Arquivo global.go

```
1   package aeinstancetest
2
3   import "google.golang.org/appengine/aetest"
4
5   var globalInstance Manager = NewManager(NewGinkgoHandler())
6
7   func CloseInstance() {
8           globalInstance.CloseInstance()
9   }
10
11  func GetInstance(opts *aetest.Options) aetest.Instance {
12          return globalInstance.GetInstance(opts)
13  }
```

Arquivo manager.go

```
1   package aeinstancetest
2
3   import (
```

```
4           "reflect"
5           "sync"
6
7           "google.golang.org/appengine/aetest"
8   )
9
10  type manager struct {
11          instance aetest.Instance
12          lock     sync.Mutex
13          opts     *aetest.Options
14          handler  Handler
15          err      error
16  }
17
18  func (h *manager) getInstanceWithoutCreate() aetest.Instance {
19          if h.err != nil {
20                  h.handler.Skip(h.err)
21          }
22          return h.instance
23  }
24
25  func (h *manager) isDifferentOptions(opts *aetest.Options) bool {
26          return !reflect.DeepEqual(opts, h.opts)
27  }
28
29  // GetInstance returns an AppEngine instance so we can test AppEngine
    ↪  services
30  func (h *manager) GetInstance(opts *aetest.Options) aetest.Instance {
31          h.lock.Lock()
32          defer h.lock.Unlock()
33          result := h.getInstanceWithoutCreate()
34          if h.err != nil {
35                  result = nil
36          }
37          if result != nil && h.isDifferentOptions(opts) {
38                  // If the options are different, we should recreate the
                    ↪  instance
39                  h.handler.Check(result.Close())
40                  result = nil
```

```
41          }
42          if result != nil || h.err != nil {
43                  return result
44          }
45          h.instance, h.err = aetest.NewInstance(opts)
46          h.opts = opts
47          return h.getInstanceWithoutCreate()
48  }
49
50  // CloseInstance just closes the appengine sdk instance opened, if there
    ↪  is one
51  func (h *manager) CloseInstance() {
52          h.lock.Lock()
53          defer h.lock.Unlock()
54          if h.instance == nil {
55                  return
56          }
57          h.handler.Check(h.instance.Close())
58          h.instance = nil
59  }
60
61  func NewManager(handler Handler) Manager {
62          return &manager{handler: handler}
63  }
```

B.1.41 Pasta robot/ddiffer/cmd/ddiffer

Arquivo main.go

```
1   package main
2
3   import (
4           "context"
5           "io"
6           "net/http"
7           "os"
8           "strings"
9           "sync"
10          "time"
11
```

```
12          "github.com/davecgh/go-spew/spew"
13          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
14          "github.com/fjorgemota/gurudamatricula/util/clock"
15          "github.com/fjorgemota/gurudamatricula/util/coder"
16          "github.com/fjorgemota/gurudamatricula/util/file"
17          "github.com/fjorgemota/gurudamatricula/util/kv"
18          "github.com/fjorgemota/gurudamatricula/util/pipeline"
19          "github.com/fjorgemota/gurudamatricula/util/taskqueue"
            ↪ //        "github.com/pkg/profile"
20          "github.com/sirupsen/logrus"
21  )
22
23  type handler struct {
24          caller taskqueue.TaskCaller
25          store  kv.Store
26          logger logrus.FieldLogger
27  }
28
29  func (h handler) GetStore(context.Context) (kv.Store, error) {
30          return h.store, nil
31  }
32  func (h handler) GetTaskQueueCaller(context.Context) taskqueue.TaskCaller
    ↪ {
33          return h.caller
34  }
35  func (h handler) GetContext(req *http.Request) context.Context {
36          return req.Context()
37  }
38  func (h handler) GetLogger(context.Context) logrus.FieldLogger {
39          return h.logger
40  }
41  func (h handler) GetManager(context.Context) (file.Manager, error) {
42          return file.NewLocalManager("data/")
43  }
44
45  func (h handler) GetStreamingDecoder(handle io.Reader)
    ↪  coder.StreamingDecoder {
46          return coder.NewJSONDecoder(handle)
47  }
```

```
func (h handler) GetStreamingEncoder(handle io.WriteCloser) coder.StreamingEncoder {
        return coder.NewJSONEncoder(handle)
}

var pipelineHandler ddiffer.PipelineHandler

var profiler interface {
        Stop()
}

func exitTask(ctx context.Context, pipe pipeline.Pipeline, result ...ddiffer.ComparerResult) error {
        spew.Dump(result)
        os.Exit(0)
        return nil
}

func finalizeConvertTask(ctx context.Context, pipe pipeline.Pipeline, result1, result2 []ddiffer.PipelineConverterResult) error {
        var results []interface{}
        var err error
        resultInputs := make(map[ddiffer.TargetType]struct {
                result1, result2 ddiffer.PipelineConverterResult
        }, len(result1))
        for _, result := range result1 {
                input := resultInputs[result.Target]
                input.result1 = result
                resultInputs[result.Target] = input
        }
        for _, result := range result2 {
                input := resultInputs[result.Target]
                input.result2 = result
                resultInputs[result.Target] = input
        }
        for target, input := range resultInputs {
                var result pipeline.FutureID
                if err == nil {
                        targetString := strings.ToLower(target.String())
```

```
84                          result, err = pipelineHandler.Compare(pipe,
                            ↪  ddiffer.ComparerOptions{
85                                  Target:           target,
86                                  OldSource:        input.result1.Result,
87                                  NewSource:        input.result2.Result,
88                                  CreatedTemplate:  "usp-result-created-" +
                                    ↪  targetString + "-part-%d.dat",
89                                  ModifiedTemplate: "usp-result-modified-"
                                    ↪  + targetString + "-part-%d.dat",
90                                  DeletedTemplate:  "usp-result-deleted-" +
                                    ↪  targetString + "-part-%d.dat",
91                          })
92                  }
93                  if err == nil {
94
95                          results = append(results, result)
96                  }
97          }
98          if err == nil {
99                  _, err = pipe.Dispatch("exit", results...)
100         }
101         // profiler.Stop()
102         return err
103 }
104 func start(ctx context.Context, pipe pipeline.Pipeline) error {
105         var result1, result2 pipeline.FutureID
106         var err error
107         result1, err = pipelineHandler.Convert(pipe,
            ↪  ddiffer.PipelineConverterOptions{
108                 UniversityID:   "usp",
109                 Source:         ddiffer.SourceHabilitations,
110                 Name:           "usp-semantic.dat",
111                 TargetTemplate: "usp-semantic-result-%s.dat",
112         })
113         if err == nil {
114                 result2, err = pipelineHandler.Convert(pipe,
                    ↪  ddiffer.PipelineConverterOptions{
115                         UniversityID:   "ufsc",
116                         Source:         ddiffer.SourceOffers,
```

```
                        Name:           "ufsc.dat",
                        TargetTemplate: "ufsc-result-%s.dat",
                })
        }
        if err == nil {
                _, err = pipe.Dispatch("finalize_convert", result1,
                ↪  result2)
        }
        return err
}
func main() {
        // profiler = profile.Start(profile.MemProfile,
        ↪  profile.ProfilePath("."))
        cod := coder.NewGobCoder()
        store := kv.NewMemoryStore()
        clk := clock.NewRealClock()
        taskDispatcher := taskqueue.NewDispatcher(cod)
        pipelineDispatcher := pipeline.NewDispatcher(cod)
        err := pipelineDispatcher.Register("exit", exitTask)
        if err == nil {
                err = pipelineDispatcher.Register("finalize_convert",
                ↪  finalizeConvertTask)
        }
        if err == nil {
                err = pipelineDispatcher.Register("start", start)
        }
        logger := logrus.New()
        logger.Level = logrus.WarnLevel
        taskCaller := taskqueue.NewInlineCaller(taskDispatcher,
        ↪  taskqueue.InlineCallerOptions{
                Concurrency: 2,
                Base:        "http://127.0.0.1:8005/",
                Logger:      logger,
                Retries:     1,
                Clock:       clk,
        })
        h := handler{
                caller: taskCaller,
                store:  store,
```

```
152                 logger: logger,
153         }
154         worker := taskqueue.NewHTTPWorker(taskDispatcher, h)
155         http.Handle("/", worker)
156         if err != nil {
157                 logger.WithError(err).Fatal("Error while starting")
158         }
159         var pipelineOptions pipeline.GlobalOptions
160         pipelineOptions.MaxAttempts = 1
161         pipeline.RegisterWorker(taskDispatcher, pipelineDispatcher, cod,
            ↪  clk, h, pipelineOptions)
162         var wg sync.WaitGroup
163         wg.Add(1)
164         go func() {
165                 logger.Debug("Starting server")
166                 errServer := http.ListenAndServe("127.0.0.1:8005",
                    ↪  http.DefaultServeMux)
167                 if errServer != nil {
168                         logger.WithError(errServer).Fatal("Error while
                            ↪  starting server")
169                 }
170                 wg.Done()
171         }()
172         time.Sleep(2 * time.Second)
173         pipe := pipeline.NewPipeline(logger, cod, clk, taskCaller, store,
            ↪  pipelineDispatcher, pipelineOptions)
174         var options ddiffer.PipelineGlobalOptions
175         options.MergeSourcesLimit = 2
176         options.ConvertersLimits.Items = 2500
177         options.ComparersLimits.Items = 2000
178         options.ComparersLimits.Size = 1024 * 1024 * 10 * 4
179         if err == nil {
180                 pipelineHandler, err =
                    ↪  ddiffer.NewPipelineHandler(pipelineDispatcher, h,
                    ↪  options)
181         }
182         if err == nil {
183                 _, err = pipe.Dispatch("start")
184         }
```

```
185         if err != nil {
186                 logger.WithError(err).Fatal("Error while starting bot")
187         }
188         wg.Wait()
189 }
```

B.1.42 Pasta robot/ufsc2offers/cmd/ufsc2offers

Arquivo main.go

```
1  package main
2
3  import (
4          "crypto/tls"
5          "fmt"
6          "io"
7          "net/http"
8          "os"
9          "path"
10         "sync"
11         "time"
12
13         "github.com/boltdb/bolt"
14         "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
15         "github.com/fjorgemota/gurudamatricula/robot/ufsc2offers"
16         "github.com/fjorgemota/gurudamatricula/robot/util/ccookiejar"
17         "github.com/fjorgemota/gurudamatricula/util/clock"
18         "github.com/fjorgemota/gurudamatricula/util/coder"
19         "github.com/fjorgemota/gurudamatricula/util/file"
20         "github.com/fjorgemota/gurudamatricula/util/kv"
21         "github.com/fjorgemota/gurudamatricula/util/pipeline"
22         "github.com/fjorgemota/gurudamatricula/util/taskqueue"
23         "github.com/fjorgemota/gurudamatricula/util/testutil"
24         "github.com/sirupsen/logrus"
25         "golang.org/x/net/context"
26 )
27
28 type botHandler struct {
29         logger    logrus.FieldLogger
30         store     kv.Store
```

```
31          caller    taskqueue.TaskCaller
32          transport http.RoundTripper
33  }
34
35  func (bh botHandler) GetTaskQueueCaller(ctx context.Context)
    ↪  taskqueue.TaskCaller {
36          return bh.caller
37  }
38
39  func (bh botHandler) GetContext(req *http.Request) context.Context {
40          return req.Context()
41  }
42
43  func (bh botHandler) GetStore(ctx context.Context) (kv.Store, error) {
44          return bh.store, nil
45  }
46
47  func (bh botHandler) GetHTTPClient(ctx context.Context, jar
    ↪  http.CookieJar) *http.Client {
48          return &http.Client{
49                  Jar:       jar,
50                  Transport: bh.transport,
51          }
52  }
53
54  func (bh botHandler) GetManager(context.Context) (file.Manager, error) {
55          current, err := os.Getwd()
56          var p string
57          if err == nil {
58                  p = path.Join(current, "data")
59          }
60          var manager file.Manager
61          if err == nil {
62                  manager, err = file.NewLocalManager(p)
63          }
64          return manager, err
65  }
66
67  func (bh botHandler) GetCookieJar() (ccookiejar.CodedCookieJar, error) {
```

```
68          return ccookiejar.NewCookieJarWithOptions(nil)
69  }
70
71  func (bh botHandler) GetLogger(context.Context) logrus.FieldLogger {
72          return bh.logger
73  }
74
75  func (bh botHandler) GetStreamingEncoder(writer io.WriteCloser)
    ↪  coder.StreamingEncoder {
76          return coder.NewJSONEncoder(writer)
77  }
78
79  func (bh botHandler) GetStreamingDecoder(reader io.Reader)
    ↪  coder.StreamingDecoder {
80          return coder.NewJSONDecoder(reader)
81  }
82
83  var bot ufsc2offers.Bot
84  var db *bolt.DB
85  var differ ddiffer.PipelineHandler
86
87  func exitTask(ctx context.Context, pipe pipeline.Pipeline, filename
    ↪  string, result []ddiffer.PipelineConverterResult) error {
88          fmt.Println("The result is in the file ", filename)
89          fmt.Println("The result of the converter: ", result)
90          var err error
91          if db != nil {
92                  err = db.Close()
93          }
94          if err == nil {
95                  os.Exit(0)
96          }
97          return err
98  }
99
100 func convert(ctx context.Context, pipe pipeline.Pipeline, filename
    ↪  string) error {
101         result, err := differ.Convert(pipe,
            ↪  ddiffer.PipelineConverterOptions{
```

```
102                 UniversityID:   "ufsc",
103                 Source:         ddiffer.SourceOffers,
104                 Name:           filename,
105                 TargetTemplate: "ufsc-offers-%s.dat",
106         })
107         if err == nil {
108                 _, err = pipe.Dispatch("exit", filename, result)
109         }
110         return err
111 }
112
113 func start(ctx context.Context, pipe pipeline.Pipeline) error {
114         result, err := bot.Start(pipe, "ufsc.dat")
115         if err == nil {
116                 _, err = pipe.Dispatch("exit", result,
                    ↪  []ddiffer.PipelineConverterResult{})
117                 // _, err = pipe.Dispatch("convert", result)
118         }
119         return err
120 }
121
122 func main() {
123         cod := coder.NewGobCoder()
124         clk := clock.NewRealClock()
125         db, err := bolt.Open("ufsc2offers.db", 0600, nil)
126         var store kv.Store
127         if err == nil {
128                 store = kv.NewBoltStore(cod, db)
129         }
130         taskDispatcher := taskqueue.NewDispatcher(cod)
131         pipelineDispatcher := pipeline.NewDispatcher(cod)
132         if err == nil {
133                 err = pipelineDispatcher.Register("exit", exitTask)
134         }
135         if err == nil {
136                 err = pipelineDispatcher.Register("convert", convert)
137         }
138         if err == nil {
139                 err = pipelineDispatcher.Register("start", start)
```

```
        }
        logger := logrus.New()
        logger.Level = logrus.DebugLevel
        if err != nil {
                logger.WithError(err).Fatal("Error while starting")
        }
        taskCaller := taskqueue.NewInlineCaller(taskDispatcher,
        ↪  taskqueue.InlineCallerOptions{
                Concurrency: 4,
                Base:        "http://127.0.0.1:8005/",
                Logger:      logger,
                Retries:     3,
                Clock:       clk,
        })
        var transport http.RoundTripper = &http.Transport{
                TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
                Proxy:           http.ProxyFromEnvironment,
        }
        mode := testutil.ModeDisabled
        if err == nil && mode != testutil.ModeDisabled {
                transport = testutil.NewRecorder(transport, store, cod,
                ↪  testutil.RecorderOptions{
                        Mode: mode,
                })
        }
        handler := &botHandler{logger: logger, store: store, caller:
        ↪  taskCaller, transport: transport}
        worker := taskqueue.NewHTTPWorker(taskDispatcher, handler)
        http.Handle("/", worker)
        var pipelineOptions pipeline.GlobalOptions
        pipeline.RegisterWorker(taskDispatcher, pipelineDispatcher, cod,
        ↪  clk, handler, pipelineOptions)
        var wg sync.WaitGroup
        wg.Add(1)
        go func() {
                logger.Debug("Starting server")
                errServer := http.ListenAndServe("127.0.0.1:8005",
                ↪  http.DefaultServeMux)
                if errServer != nil {
```

```
174                         logger.WithError(errServer).Fatal("Error while
                            ↪  starting server")
175                 }
176                 wg.Done()
177         }()
178         time.Sleep(2 * time.Second)
179         pipe := pipeline.NewPipeline(logger, cod, clk, taskCaller, store,
            ↪  pipelineDispatcher, pipelineOptions)
180         var options ufsc2offers.Options
181         options.NumberOfPeriods = 3
182         options.DelayBetweenRequests = 1 * time.Second
183         if mode == testutil.ModeReplay {
184                 options.DelayBetweenRequests = 1 * time.Nanosecond
185         }
186         options.CheckTable = "ufsc_checks"
187         if err == nil {
188                 bot, err = ufsc2offers.NewBot(pipelineDispatcher, cod,
                    ↪  clk, handler, options)
189         }
190         if err == nil {
191                 differ, err =
                    ↪  ddiffer.NewPipelineHandler(pipelineDispatcher,
                    ↪  handler, ddiffer.PipelineGlobalOptions{})
192         }
193         if err == nil {
194                 _, err = pipe.Dispatch("start")
195         }
196         if err != nil {
197                 logger.WithError(err).Fatal("Error while starting bot")
198         }
199         wg.Wait()
200 }
```

B.1.43 Pasta robot/usp2habilitations/cmd/usp2habilitations

Arquivo main.go

```
1 package main
2
3 import (
```

```
4           "crypto/tls"
5           "fmt"
6           "io"
7           "math/rand"
8           "net/http"
9           "os"
10          "path"
11          "sync"
12          "time"
13
14          "github.com/fjorgemota/gurudamatricula/robot/usp2habilitations"
15
16          "github.com/boltdb/bolt"
17          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
18          "github.com/fjorgemota/gurudamatricula/util/clock"
19          "github.com/fjorgemota/gurudamatricula/util/coder"
20          "github.com/fjorgemota/gurudamatricula/util/file"
21          "github.com/fjorgemota/gurudamatricula/util/kv"
22          "github.com/fjorgemota/gurudamatricula/util/pipeline"
23          "github.com/fjorgemota/gurudamatricula/util/taskqueue"
24          "github.com/fjorgemota/gurudamatricula/util/testutil"
25          "github.com/sirupsen/logrus"
26          "golang.org/x/net/context"
27  )
28
29  type botHandler struct {
30          logger logrus.FieldLogger
31          store  kv.Store
32          caller taskqueue.TaskCaller
33          client *http.Client
34  }
35
36  func (bh botHandler) GetTaskQueueCaller(ctx context.Context)
    ↪  taskqueue.TaskCaller {
37          return bh.caller
38  }
39
40  func (bh botHandler) GetContext(req *http.Request) context.Context {
41          return req.Context()
```

```
42 }
43
44 func (bh botHandler) GetStore(ctx context.Context) (kv.Store, error) {
45         return bh.store, nil
46 }
47
48 func (bh botHandler) GetHTTPClient(ctx context.Context, _ http.CookieJar)
   ↪  *http.Client {
49         return bh.client
50 }
51
52 func (bh botHandler) GetManager(context.Context) (file.Manager, error) {
53         current, err := os.Getwd()
54         var p string
55         if err == nil {
56                 p = path.Join(current, "data")
57         }
58         var manager file.Manager
59         if err == nil {
60                 manager, err = file.NewLocalManager(p)
61         }
62         return manager, err
63 }
64
65 func (bh botHandler) GetLogger(context.Context) logrus.FieldLogger {
66         return bh.logger
67 }
68
69 func (bh botHandler) GetStreamingEncoder(writer io.WriteCloser)
   ↪  coder.StreamingEncoder {
70         return coder.NewJSONEncoder(writer)
71 }
72
73 func (bh botHandler) GetStreamingDecoder(reader io.Reader)
   ↪  coder.StreamingDecoder {
74         return coder.NewJSONDecoder(reader)
75 }
76
77 var bot usp2habilitations.Bot
```

```
78 var db *bolt.DB
79 var differ ddiffer.PipelineHandler
80
81 func exitTask(ctx context.Context, pipe pipeline.Pipeline, filename
   ↪  string, result []ddiffer.PipelineConverterResult) error {
82         fmt.Println("The result is in the file ", filename)
83         fmt.Println("The result of the converter: ", result)
84         var err error
85         if db != nil {
86                 err = db.Close()
87         }
88         if err == nil {
89                 os.Exit(0)
90         }
91         return err
92 }
93
94 func convert(ctx context.Context, pipe pipeline.Pipeline, filename
   ↪  string) error {
95         result, err := differ.Convert(pipe,
           ↪  ddiffer.PipelineConverterOptions{
96                 UniversityID:   "usp",
97                 Source:         ddiffer.SourceHabilitations,
98                 Name:           filename,
99                 TargetTemplate: "usp-habilitations-%s.dat",
100        })
101        if err == nil {
102                _, err = pipe.Dispatch("exit", filename, result)
103        }
104        return err
105 }
106
107 func start(ctx context.Context, pipe pipeline.Pipeline) error {
108        result, err := bot.Start(pipe, "usp-semantic.dat")
109        if err == nil {
110                _, err = pipe.Dispatch("exit", result,
                   ↪  []ddiffer.PipelineConverterResult{})
111                // _, err = pipe.Dispatch("convert", result)
112        }
```

```
113         return err
114 }
115
116 func main() {
117         rand.Seed(time.Now().UnixNano())
118         db, err := bolt.Open("usp2habilitations.db", 0600, nil)
119         cod := coder.NewGobCoder()
120         clk := clock.NewRealClock()
121         var store kv.Store
122         if err == nil {
123                 store = kv.NewBoltStore(cod, db)
124         }
125         var tr http.RoundTripper = &http.Transport{
126                 TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
127                 Proxy:           http.ProxyFromEnvironment,
128         }
129         mode := testutil.ModeReplay
130         if err == nil && mode != testutil.ModeDisabled {
131                 tr = testutil.NewRecorder(tr, store, cod,
                  ↪  testutil.RecorderOptions{
132                         Mode:     mode,
133                         MatchAll: true,
134                 })
135         }
136         client := &http.Client{
137                 Transport: tr,
138                 Timeout:   5 * time.Minute,
139         }
140         taskDispatcher := taskqueue.NewDispatcher(cod)
141         pipelineDispatcher := pipeline.NewDispatcher(cod)
142         if err == nil {
143                 err = pipelineDispatcher.Register("exit", exitTask)
144         }
145         if err == nil {
146                 err = pipelineDispatcher.Register("convert", convert)
147         }
148         if err == nil {
149                 err = pipelineDispatcher.Register("start", start)
150         }
```

```
151         logger := logrus.New()
152         // logger.Out = ioutil.Discard
153         logger.Level = logrus.DebugLevel
154         if err != nil {
155                 logger.WithError(err).Fatal("Error while starting")
156         }
157         taskCaller := taskqueue.NewInlineCaller(taskDispatcher,
            ↪  taskqueue.InlineCallerOptions{
158                 Retries:     3,
159                 Logger:      logger,
160                 Base:        "http://127.0.0.1:8005/",
161                 Clock:       clk,
162                 Concurrency: 4,
163         })
164         handler := &botHandler{logger, store, taskCaller, client}
165         worker := taskqueue.NewHTTPWorker(taskDispatcher, handler)
166         http.Handle("/", worker)
167         var pipelineOptions pipeline.GlobalOptions
168         if err == nil {
169                 err = pipeline.RegisterWorker(taskDispatcher,
                  ↪  pipelineDispatcher, cod, clk, handler,
                  ↪  pipelineOptions)
170         }
171         var wg sync.WaitGroup
172         wg.Add(1)
173         go func() {
174                 logger.Debug("Starting server")
175                 errServer := http.ListenAndServe("127.0.0.1:8005",
                  ↪  http.DefaultServeMux)
176                 if errServer != nil {
177                         logger.WithError(errServer).Fatal("Error while
                          ↪  starting server")
178                 }
179                 wg.Done()
180         }()
181         time.Sleep(2 * time.Second)
182         pipe := pipeline.NewPipeline(logger, cod, clk, taskCaller, store,
            ↪  pipelineDispatcher, pipelineOptions)
183         var options usp2habilitations.PipelineOptions
```

```
184         options.ParallelRequests = 4
185         options.Crawler.Base.DelayBetweenRequests = 1 * time.Second
186         if mode == testutil.ModeReplay {
187                 options.ParallelRequests = 1024
188                 options.Crawler.Base.DelayBetweenRequests = 1 *
                    ↪ time.Nanosecond
189         }
190         options.Crawler.GetDisciplines = true
191         if err == nil {
192                 differ, err =
                    ↪ ddiffer.NewPipelineHandler(pipelineDispatcher,
                    ↪ handler, ddiffer.PipelineGlobalOptions{})
193         }
194         if err == nil {
195                 bot, err = usp2habilitations.NewBot(pipelineDispatcher,
                    ↪ clk, handler, options)
196         }
197         if err == nil {
198                 _, err = pipe.Dispatch("start")
199         }
200         if err != nil {
201                 logger.WithError(err).Fatal("Error while starting bot")
202         }
203         wg.Wait()
204 }
```

B.1.44 Pasta robot/usp2offers/cmd/usp2offers

Arquivo main.go

```
1  package main
2
3  import (
4          "crypto/tls"
5          "fmt"
6          "io"
7          "net/http"
8          "os"
9          "path"
10         "sync"
```

```
11          "time"
12
13          "github.com/boltdb/bolt"
14          "github.com/fjorgemota/gurudamatricula/robot/ddiffer"
15          "github.com/fjorgemota/gurudamatricula/robot/usp2offers"
16          "github.com/fjorgemota/gurudamatricula/util/clock"
17          "github.com/fjorgemota/gurudamatricula/util/coder"
18          "github.com/fjorgemota/gurudamatricula/util/file"
19          "github.com/fjorgemota/gurudamatricula/util/kv"
20          "github.com/fjorgemota/gurudamatricula/util/pipeline"
21          "github.com/fjorgemota/gurudamatricula/util/taskqueue"
22          "github.com/fjorgemota/gurudamatricula/util/testutil"
23          "github.com/sirupsen/logrus"
24          "golang.org/x/net/context"
25  )
26
27  type botHandler struct {
28          logger  logrus.FieldLogger
29          store   kv.Store
30          caller  taskqueue.TaskCaller
31          client  *http.Client
32          manager file.Manager
33  }
34
35  func (bh botHandler) GetTaskQueueCaller(ctx context.Context)
    ↪  taskqueue.TaskCaller {
36          return bh.caller
37  }
38
39  func (bh botHandler) GetStore(ctx context.Context) (kv.Store, error) {
40          return bh.store, nil
41  }
42
43  func (bh botHandler) GetContext(req *http.Request) context.Context {
44          return req.Context()
45  }
46
47  func (bh botHandler) GetHTTPClient(ctx context.Context, _ http.CookieJar)
    ↪  *http.Client {
```

```
48          return bh.client
49  }
50
51  func (bh botHandler) GetManager(context.Context) (file.Manager, error) {
52          var err error
53          if bh.manager == nil {
54                  current, err := os.Getwd()
55                  var p string
56                  if err == nil {
57                          p = path.Join(current, "data")
58                  }
59                  if err == nil {
60                          bh.manager, err = file.NewLocalManager(p)
61                  }
62          }
63          return bh.manager, err
64  }
65
66  func (bh botHandler) GetLogger(context.Context) logrus.FieldLogger {
67          return bh.logger
68  }
69
70  func (bh botHandler) GetStreamingEncoder(writer io.WriteCloser)
    ↪  coder.StreamingEncoder {
71          return coder.NewJSONEncoder(writer)
72  }
73
74  func (bh botHandler) GetStreamingDecoder(reader io.Reader)
    ↪  coder.StreamingDecoder {
75          return coder.NewJSONDecoder(reader)
76  }
77
78  var bot usp2offers.Bot
79  var differ ddiffer.PipelineHandler
80  var db *bolt.DB
81
82  func exitTask(ctx context.Context, pipe pipeline.Pipeline, filename
    ↪  string, result []ddiffer.PipelineConverterResult) error {
83          fmt.Println("The result is in the file ", filename)
```

```
84          fmt.Println("The result of the converter: ", result)
85          var err error
86          if db != nil {
87                  err = db.Close()
88          }
89          if err == nil {
90                  os.Exit(0)
91          }
92          return err
93  }
94
95  func convert(ctx context.Context, pipe pipeline.Pipeline, filename
    ↪  string) error {
96          result, err := differ.Convert(pipe,
            ↪  ddiffer.PipelineConverterOptions{
97                  UniversityID:   "usp",
98                  Source:         ddiffer.SourceOffers,
99                  Name:           filename,
100                 TargetTemplate: "usp-offers-%s.dat",
101         })
102         if err == nil {
103                 _, err = pipe.Dispatch("exit", filename, result)
104         }
105         return err
106 }
107
108 func start(ctx context.Context, pipe pipeline.Pipeline) error {
109         result, err := bot.Start(pipe, "usp.dat")
110         if err == nil {
111                 _, err = pipe.Dispatch("exit", result,
                    ↪  []ddiffer.PipelineConverterResult{})
112                 // _, err = pipe.Dispatch("convert", result)
113         }
114         return err
115 }
116 func main() {
117         cod := coder.NewGobCoder()
118         clk := clock.NewRealClock()
119         db, err := bolt.Open("usp2offers.db", 0600, nil)
```

```
120         var store kv.Store
121         if err == nil {
122                 store = kv.NewBoltStore(cod, db)
123         }
124         var tr http.RoundTripper = &http.Transport{
125                 TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
126                 Proxy:           http.ProxyFromEnvironment,
127         }
128         mode := testutil.ModeDisabled
129         if err == nil && mode != testutil.ModeDisabled {
130                 tr = testutil.NewRecorder(tr, store, cod,
                    ↪  testutil.RecorderOptions{
131                         Mode:     mode,
132                         MatchAll: true,
133                 })
134         }
135         client := &http.Client{
136                 Transport: tr,
137                 Timeout:   5 * time.Minute,
138         }
139         taskDispatcher := taskqueue.NewDispatcher(cod)
140         pipelineDispatcher := pipeline.NewDispatcher(cod)
141         if err == nil {
142                 err = pipelineDispatcher.Register("exit", exitTask)
143         }
144         if err == nil {
145                 err = pipelineDispatcher.Register("convert", convert)
146         }
147         if err == nil {
148                 err = pipelineDispatcher.Register("start", start)
149         }
150         logger := logrus.New()
151         logger.Level = logrus.DebugLevel
152         if err != nil {
153                 logger.WithError(err).Fatal("Error while starting")
154         }
155         taskCaller := taskqueue.NewInlineCaller(taskDispatcher,
          ↪  taskqueue.InlineCallerOptions{
156                 Retries:     3,
```

```
157                 Base:        "http://127.0.0.1:8015/",
158                 Logger:      logger,
159                 Clock:       clk,
160                 Concurrency: 4,
161         })
162         handler := &botHandler{logger: logger, store: store, caller:
            ↪  taskCaller, client: client, manager: nil}
163         worker := taskqueue.NewHTTPWorker(taskDispatcher, handler)
164         http.Handle("/", worker)
165         var pipelineOptions pipeline.GlobalOptions
166         pipeline.RegisterWorker(taskDispatcher, pipelineDispatcher, cod,
            ↪  clk, handler, pipelineOptions)
167         var wg sync.WaitGroup
168         wg.Add(1)
169         go func() {
170                 logger.Debug("Starting server")
171                 errServer := http.ListenAndServe("127.0.0.1:8015",
                    ↪  http.DefaultServeMux)
172                 if errServer != nil {
173                         logger.WithError(errServer).Fatal("Error while
                            ↪  starting server")
174                 }
175                 wg.Done()
176         }()
177         time.Sleep(2 * time.Second)
178         pipe := pipeline.NewPipeline(logger, cod, clk, taskCaller, store,
            ↪  pipelineDispatcher, pipelineOptions)
179         var options usp2offers.Options
180         options.ParallelRequests = 4
181         options.DelayBetweenRequests = 1 * time.Second
182         if mode == testutil.ModeReplay {
183                 options.DelayBetweenRequests = 1 * time.Nanosecond
184                 options.ParallelRequests = 1024
185         }
186         differ, err = ddiffer.NewPipelineHandler(pipelineDispatcher,
            ↪  handler, ddiffer.PipelineGlobalOptions{})
187         if err == nil {
188                 bot, err = usp2offers.NewBot(pipelineDispatcher, clk,
                    ↪  handler, options)
```

```
189         }
190         if err == nil {
191                 _, err = pipe.Dispatch("start")
192         }
193         if err != nil {
194                 logger.WithError(err).Fatal("Error while starting bot")
195         }
196         wg.Wait()
197 }
```

B.1.45 Pasta robot/util/uspreq/cmd/uspreq

Arquivo main.go

```
1  package main
2
3  import (
4          "crypto/tls"
5          "io"
6          "net/http"
7          "os"
8          "time"
9
10         "github.com/boltdb/bolt"
11         "github.com/fjorgemota/gurudamatricula/robot/format"
12         "github.com/fjorgemota/gurudamatricula/robot/util/uspreq"
13         "github.com/fjorgemota/gurudamatricula/util/coder"
14         "github.com/fjorgemota/gurudamatricula/util/kv"
15         "github.com/fjorgemota/gurudamatricula/util/testutil"
16         "github.com/sirupsen/logrus"
17 )
18
19 func main() {
20         var db *bolt.DB
21         cod := coder.NewGobCoder()
22         var tr http.RoundTripper = &http.Transport{
23                 TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
24                 Proxy:           http.ProxyFromEnvironment,
25         }
26         var err error
```

```
27          if err == nil {
28                  db, err = bolt.Open("usp.db", 0600, &bolt.Options{
29                          ReadOnly: false,
30                  })
31          }
32          if err == nil && db != nil {
33                  recStore := kv.NewBoltStore(cod, db)
34                  tr = testutil.NewRecorder(tr, recStore, cod,
                    ↪  testutil.RecorderOptions{
35                          Mode:     testutil.ModeReplay |
                            ↪  testutil.ModeRecord,
36                          MatchAll: true,
37                  })
38          }
39          client := &http.Client{
40                  Transport: tr,
41                  Timeout:   5 * time.Minute,
42          }
43          logger := logrus.New()
44          logger.Level = logrus.DebugLevel
45          var campi []format.Campus
46          if err == nil {
47                  campi, err = uspreq.GetCampi(client, "xpto", "Mozilla/5.0
                    ↪  (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
                    ↪  Gecko) Chrome/60.0.3112.101 Safari/537.36")
48          }
49          var file *os.File
50          if err == nil {
51                  file, err = os.Create("campi.dat")
52          }
53          var data io.Reader
54          if err == nil {
55                  data, err = cod.ConvertToReader(&campi)
56          }
57          if err == nil {
58                  _, err = io.Copy(file, data)
59          }
60          if err == nil {
61                  err = file.Close()
```

```
62          }
63          if err != nil {
64                  logger.WithError(err).Fatal("Error while starting")
65          }
66
67  }
```

B.2 Frontend

Arquivo .babelrc

```
1  {
2    "presets": [
3      [
4        "@babel/preset-env",
5        {
6          "modules": false
7        }
8      ]
9    ],
10   "plugins": [
11     "@babel/plugin-syntax-dynamic-import",
12     "babel-plugin-graphql-tag",
13     "@babel/plugin-transform-react-constant-elements",
14     [
15       "@babel/plugin-transform-react-jsx",
16       {
17         "pragma": "h",
18         "useBuiltIns": true
19       }
20     ]
21   ]
22 }
```

Arquivo .browserslistrc

```
1  last 2 Chrome versions
2  last 2 ChromeAndroid versions
3  last 2 Firefox versions
```

```
last 2 FirefoxAndroid versions
last 2 Safari versions
last 2 iOS versions
last 2 Edge versions
last 2 Opera versions
```

Arquivo .eslintrc.json

```
{
    "env": {
        "browser": true,
        "es6": true
    },
    "extends": "eslint:recommended",
    "globals": {
        "Atomics": "readonly",
        "SharedArrayBuffer": "readonly"
    },
    "parserOptions": {
        "ecmaFeatures": {
            "jsx": true
        },
        "ecmaVersion": 2018,
        "sourceType": "module"
    },
    "plugins": [
        "react"
    ],
    "rules": {
    }
}
```

Arquivo codegen.yml

```
overwrite: true
schema: "src/js/schema.graphql"
documents: src/js/**/*.gql
config:
  scalars:
```

```
    Time: Date
  typesPrefix: Remote
  skipTypename: true
  maybeValue: T | undefined
  constEnums: true
generates:
  src/js/graphql.ts:
    plugins:
      - typescript
      - typescript-resolvers
      - typescript-operations
  src/js/index.d.ts:
    plugins:
      - typescript-graphql-files-modules
```

Arquivo config.toml

```
baseURL = "http://example.org/"
languageCode = "pt-br"
title = "Guru da Matrícula"
[outputs]
    home = ["html", "json", "rss"]
    taxonomy = ["html", "json", "rss"]
    taxonomyTerm = ["html", "json", "rss"]
    section = ["html", "json", "rss"]
    page = ["html", "json"]

[related]
  threshold = 10
  includeNewer = true
  toLower = true


[[related.indices]]
  name= "tags"
  weight= 100

[[related.indices]]
  name= "categories"
  weight= 80
```

```
24
25
26 [[related.indices]]
27   name= "type"
28   weight= 50
29
30 [[related.indices]]
31   name = "date"
32   weight= 10
```

Arquivo package.json

```
1  {
2    "name": "gurudamatricula-frontend",
3    "version": "0.1.0",
4    "description": "Frontend of the Guru da Matricula project",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "prod:watch": "webpack-dashboard -- ./node_modules/.bin/webpack
       ↪ --mode=production --watch",
9      "prod:build": "./node_modules/.bin/webpack --mode=production",
10     "prod:analyze": "webpack --mode=production --profile --stats --json >
       ↪  stats.json && webpack-bundle-analyzer stats.json static/static/",
11     "dev:watch": "webpack-dashboard -- ./node_modules/.bin/webpack
       ↪ --mode=development --watch",
12     "dev:build": "webpack-dashboard -- ./node_modules/.bin/webpack
       ↪ --mode=development",
13     "dev:analyze": "webpack --mode=development --profile --stats --json >
       ↪  stats.json && webpack-bundle-analyzer stats.json static/static/",
14     "postinstall": "patch-package",
15     "storybook": "start-storybook -p 9001 -c .storybook",
16     "generate": "graphql-codegen --config codegen.yml"
17   },
18   "author": "Fernando Jorge Mota <contato@fjorgemota.com>",
19   "license": "MIT",
20   "private": true,
21   "devDependencies": {
22     "@babel/core": "^7.7.2",
23     "@babel/plugin-syntax-dynamic-import": "^7.2.0",
```

```
24      "@babel/plugin-transform-react-constant-elements": "^7.2.0",
25      "@babel/plugin-transform-react-jsx": "^7.7.0",
26      "@babel/preset-env": "^7.7.1",
27      "@graphql-codegen/cli": "^1.9.0",
28      "@graphql-codegen/introspection": "^1.9.0",
29      "@graphql-codegen/typescript": "1.9.1",
30      "@graphql-codegen/typescript-graphql-files-modules": "^1.9.0",
31      "@graphql-codegen/typescript-operations": "1.9.1",
32      "@graphql-codegen/typescript-resolvers": "^1.9.0",
33      "@storybook/addon-viewport": "^5.2.6",
34      "@storybook/react": "^5.2.6",
35      "@types/benchmark": "^1.0.31",
36      "@types/chai": "^4.2.5",
37      "@types/cuid": "^1.3.0",
38      "@types/graphql": "^14.5.0",
39      "@types/history": "^4.7.3",
40      "@types/less": "^3.0.1",
41      "@types/lunr": "^2.3.2",
42      "@types/mocha": "^5.2.7",
43      "@types/react": "16.9.15",
44      "@types/react-dom": "16.9.4",
45      "@types/react-loadable": "^5.5.1",
46      "@types/react-redux": "^7.1.5",
47      "@types/react-router": "^5.1.3",
48      "@types/react-router-dom": "^5.1.3",
49      "@types/redux": "^3.6.0",
50      "@types/redux-saga": "^0.10.5",
51      "@types/reselect": "^2.2.0",
52      "@types/typescript": "^2.0.0",
53      "@types/webpack": "^4.41.0",
54      "@types/webpack-env": "^1.14.0",
55      "@typescript-eslint/parser": "^2.9.0",
56      "awesome-typescript-loader": "^5.2.1",
57      "babel-loader": "^8.0.6",
58      "babel-plugin-graphql-tag": "^2.5.0",
59      "babel-plugin-module-resolver": "^3.2.0",
60      "babel-plugin-transform-react-remove-prop-types": "^0.4.24",
61      "benchmark": "^2.1.4",
62      "chai": "^4.2.0",
```

```
63      "css-loader": "^3.1.0",
64      "csso-webpack-plugin": "^1.0.0-beta.12",
65      "csstype": "^2.6.7",
66      "eslint": "^6.6.0",
67      "eslint-config-standard": "^14.1.0",
68      "eslint-plugin-graphql": "^3.1.0",
69      "eslint-plugin-import": "^2.18.2",
70      "eslint-plugin-jsx-a11y": "^6.2.3",
71      "eslint-plugin-node": "^10.0.0",
72      "eslint-plugin-promise": "^4.1.1",
73      "eslint-plugin-react": "^7.16.0",
74      "eslint-plugin-react-hooks": "^2.3.0",
75      "eslint-plugin-standard": "^4.0.1",
76      "file-loader": "^5.0.2",
77      "karma": "^4.4.1",
78      "karma-chrome-launcher": "^3.1.0",
79      "karma-firefox-launcher": "^1.2.0",
80      "karma-webpack": "^4.0.1",
81      "less": "^3.10.3",
82      "media-query-plugin": "^1.3.1",
83      "mini-css-extract-plugin": "^0.8.0",
84      "mocha": "^6.2.2",
85      "node-sass": "^4.13.0",
86      "null-loader": "^3.0.0",
87      "optimize-css-assets-webpack-plugin": "^5.0.3",
88      "patch-package": "^6.2.0",
89      "quicktype": "^15.0.207",
90      "sass-loader": "^8.0.0",
91      "size-limit": "^2.2.1",
92      "size-plugin": "^2.0.0",
93      "style-loader": "^1.0.0",
94      "terser-webpack-plugin": "^2.2.1",
95      "ts-loader": "^6.2.1",
96      "tsconfig-paths-webpack-plugin": "^3.2.0",
97      "typescript": "^3.7.2",
98      "url-loader": "^3.0.0",
99      "webpack": "^4.41.2",
100     "webpack-bundle-analyzer": "^3.6.0",
101     "webpack-cli": "^3.3.10",
```

```
    "webpack-dashboard": "^3.2.0",
    "webpack-dev-server": "^3.9.0",
    "webpack-modules": "^1.0.0",
    "webpack-stats-plugin": "^0.3.0",
    "workbox-webpack-plugin": "^5.0.0-rc.0",
    "worker-loader": "^2.0.0"
  },
  "dependencies": {
    "@apollo/react-hooks": "^3.1.3",
    "@material/animation": "^3.1.0",
    "@material/elevation": "^3.1.0",
    "@material/ripple": "^3.2.0",
    "@material/shape": "^3.1.0",
    "@reduxjs/toolkit": "^1.0.4",
    "@rmwc/button": "^5.7.0",
    "@rmwc/card": "^5.7.0",
    "@rmwc/checkbox": "^5.7.0",
    "@rmwc/chip": "^5.7.0",
    "@rmwc/data-table": "^5.7.0",
    "@rmwc/dialog": "^5.7.0",
    "@rmwc/drawer": "^5.7.0",
    "@rmwc/grid": "^5.7.0",
    "@rmwc/icon": "^5.7.0",
    "@rmwc/icon-button": "^5.7.0",
    "@rmwc/linear-progress": "^5.7.0",
    "@rmwc/list": "^5.7.0",
    "@rmwc/ripple": "^5.7.0",
    "@rmwc/select": "^5.7.1",
    "@rmwc/snackbar": "^5.7.0",
    "@rmwc/switch": "^5.7.0",
    "@rmwc/tabs": "^5.7.0",
    "@rmwc/textfield": "^5.7.0",
    "@rmwc/top-app-bar": "^5.7.0",
    "@rmwc/typography": "^5.7.1",
    "apollo-cache-inmemory": "^1.6.3",
    "apollo-client": "^2.6.4",
    "apollo-link": "^1.2.12",
    "apollo-link-context": "^1.0.19",
    "apollo-link-error": "^1.1.11",
```

```
141     "apollo-link-http": "^1.5.15",
142     "bind-decorator": "^1.0.11",
143     "connected-react-router": "^6.6.0",
144     "cuid": "^2.1.6",
145     "graphql": "^14.5.7",
146     "graphql-tag": "^2.10.1",
147     "graphql-voyager": "^1.0.0-rc.27",
148     "graphql.js": "^0.6.6",
149     "hashids": "^2.1.0",
150     "history": "^4.10.1",
151     "idb": "^4.0.3",
152     "immer": "^5.0.0",
153     "lunr": "^2.3.8",
154     "material-design-icons": "^3.0.1",
155     "mxgraph": "^4.0.6",
156     "normalize.css": "^8.0.1",
157     "pathfinding": "^0.4.18",
158     "preact": "^10.0.5",
159     "raw-loader": "^4.0.0",
160     "react": "^16.12.0",
161     "react-dom": "^16.12.0",
162     "react-redux": "^7.1.3",
163     "react-router": "^5.0.1",
164     "react-router-dom": "^5.0.1",
165     "react-virtualized-auto-sizer": "^1.0.2",
166     "react-window": "^1.8.5",
167     "react-window-infinite-loader": "^1.0.5",
168     "redux": "^4.0.4",
169     "redux-dynamic-modules": "^5.2.0",
170     "redux-dynamic-modules-core": "^5.2.0",
171     "redux-dynamic-modules-saga": "^5.2.0",
172     "redux-optimist": "^1.0.0",
173     "redux-saga": "^1.1.3",
174     "reselect": "^4.0.0",
175     "throttle-debounce": "^2.1.0",
176     "workbox-background-sync": "^4.3.1",
177     "workbox-core": "^4.3.1",
178     "workbox-precaching": "^4.3.1",
179     "workbox-routing": "^4.3.1",
```

```
    "workbox-sw": "^4.3.1",
    "workbox-window": "^4.3.1"
  },
  "resolutions": {
    "**/@types/react": "16.9.13",
    "**/@types/react-router": "5.1.3",
    "**/redux": "^4.0.4"
  }
}
```

Arquivo tsconfig.json

```
{
  "compilerOptions": {
    "target": "es2017",
    "module": "esnext",
    "moduleResolution": "node",
    "lib": ["es6", "es2017", "dom", "dom.iterable"],
    "noImplicitAny": false,
    "strict": true,
    "jsx": "react",
    "experimentalDecorators": true,
    "noImplicitReturns": true,
    "noImplicitThis": true,
    "noFallthroughCasesInSwitch": true,
    "allowSyntheticDefaultImports": true
  },
  "include": ["src"],
  "exclude": ["node_modules", "src/js/service-worker/"],
  "compileOnSave": false
}
```

Arquivo tsconfig.worker.json

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "types": [],
    "lib": ["es6", "es2017", "esnext.asynciterable", "webworker"]
```

```
  },
  "include": [
    "src/js/service-worker/",
    "src/js/custom.d.ts",
    "src/js/schema.graphql"
  ]
}
```

Arquivo webpack.config.js

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
const path = require("path");
const fs = require("fs");
const CssoWebpackPlugin = require("csso-webpack-plugin").default;
const {
  CheckerPlugin,
  TsConfigPathsPlugin
} = require("awesome-typescript-loader");
const { StatsWriterPlugin } = require("webpack-stats-plugin");
const TerserPlugin = require("terser-webpack-plugin");
const SizePlugin = require("size-plugin");
const WebpackModules = require("webpack-modules");
var DashboardPlugin = require("webpack-dashboard/plugin");
const { InjectManifest } = require("workbox-webpack-plugin");

const srcDir = path.resolve(__dirname, "src");
const nodeModulesDir = path.resolve(__dirname, "node_modules");

let nameCache = {};
module.exports = function(env, argv) {
  const isQuiet = argv.stats && argv.json && argv.profile;
  const isProduction = argv.mode === "production";
  let name = isProduction ? "[name].[hash]" : "[name]";
  const plugins = [
    new WebpackModules(),
    new StatsWriterPlugin({
      filename: "stats.json" // Used by Hugo to see what are the
      ↪  available assets
    }),
    new CheckerPlugin(),
```

```
    new DashboardPlugin(),
    new InjectManifest({
      swSrc: "./src/js/service-worker/index.ts",
      swDest: "../sw.js"
      // importWorkboxFrom: "local"
    })
  ];
  const babelConfig = JSON.parse(fs.readFileSync(".babelrc"));
  if (isProduction) {
    plugins.push(
      new MiniCssExtractPlugin({
        filename: name + ".css",
        chunkFilename: "[name].[hash].[id].css"
      })
    );
    plugins.push(new CssoWebpackPlugin());
    babelConfig.plugins.push("transform-react-remove-prop-types");
  }
  if (!isQuiet) {
    plugins.push(new SizePlugin());
  }
  let tsConfig =
  ↪ JSON.parse(fs.readFileSync("tsconfig.json")).compilerOptions;
  let rules = [
    {
      test: /\.gql$/,
      exclude: /node_modules/,
      loader: "graphql-tag/loader"
    },
    {
      test: /\.graphql$/,
      exclude: /node_modules/,
      loader: "raw-loader"
    },
    {
      test: /\.(js|ts|tsx|js|jsx)$/,
      include: [srcDir],
      loader: "babel-loader",
      options: babelConfig
```

```
68      },
69      {
70        test: /\.tsx?$/,
71        include: [srcDir, nodeModulesDir],
72        loader: "awesome-typescript-loader",
73        options: {
74          silent: isQuiet,
75          useCache: false,
76          compilerOptions: {
77            ...tsConfig,
78            lib: ["es6", "es2017", "dom", "dom.iterable", "webworker"]
79          }
80        }
81      },
82      {
83        test: /\.s?css$/,
84        use: [
85          isProduction ? MiniCssExtractPlugin.loader : "style-loader",
86          "css-loader",
87          {
88            loader: "sass-loader", // compiles Sass to CSS
89            options: {
90              sassOptions: {
91                includePaths: [nodeModulesDir]
92              }
93            }
94          }
95        ]
96      },
97      {
98        test: /\.(ttf|eot|woff|woff2)$/,
99        use: {
100         loader: "file-loader",
101         options: {
102           name: "fonts/[name].[ext]"
103         }
104       }
105     }
106   ];
```

```
107   if (isProduction) {
108     rules.push({
109       test: path.resolve(nodeModulesDir, "preact/debug/"),
110       use: "null-loader"
111     });
112   }
113   return {
114     entry: {
115       main: "js/app.tsx"
116       // sw: "js/service-worker/index.ts"
117     },
118     output: {
119       path: path.resolve(__dirname, "static/static/"),
120       publicPath: "/static/",
121       filename: name + ".js",
122       chunkFilename: name + ".bundle.js"
123     },
124     module: {
125       rules
126     },
127     optimization: {
128       minimizer: isProduction
129         ? [
130             new TerserPlugin({
131               extractComments: true,
132               parallel: false
133               // terserOptions: {
134               //   mangle: {
135               //     properties: {
136               //       regex: /^(.*[^_]_)|(_[^_].*)$/,
137               //       keep_quoted: true
138               //     }
139               //   },
140               //   nameCache
141               // }
142             })
143           ]
144         : []
145     },
```

```
146     performance: {
147       maxEntrypointSize: 100000,
148       // Disable hints on development because we have inline source maps
          ↪  and eval
149       // which increases the bundle size a lot
150       hints: isProduction ? "warning" : false
151     },
152     stats: {
153       // Display bailout reasons
154       optimizationBailout: true
155     },
156     plugins,
157     resolve: {
158       alias: {
159         // react: "preact/compat",
160         // "react-dom": "preact/compat"
161       },
162       modules: [srcDir, "./node_modules"],
163       extensions: [".ts", ".tsx", ".js", ".json"],
164       mainFields: ["module", "main"],
165       plugins: [new TsConfigPathsPlugin()]
166     }
167   };
168 };
```

B.2.1 Pasta .storybook

Arquivo addons.js

```
1 import '@storybook/addon-viewport/register';
```

Arquivo config.js

```
1 import { configure, addDecorator } from '@storybook/preact';
2
3
4 const req = require.context('../src/js/components', true,
  ↪  /\.stories\.tsx$/);
5
6 function loadStories() {
```

```
  req.keys().forEach(filename => req(filename));
}

configure(loadStories, module);
```

Arquivo webpack.config.js

```
const path = require('path');
const { CheckerPlugin } = require('awesome-typescript-loader')

const srcDir = path.resolve(__dirname, '..', 'src');
const nodeModulesDir = path.resolve(__dirname, '..', 'node_modules');

module.exports = {
    module: {
        rules: [
                {
                test: /\.(js|tsx?|scss)$/,
                use: 'cache-loader',
                include: [
                    srcDir,
                ]
            },
            {
                test: /\.(js|ts|tsx|js|jsx)$/,
                include: [
                    srcDir,
                ],
                use: {
                    loader: 'babel-loader'
                }
            },
            {
                test: /\.tsx?$/,
                include: [
                    srcDir,
                    nodeModulesDir,
                ],
                loader: 'awesome-typescript-loader',
                options: {
```

```
                    useCache: true
                }
            },
            {
                test: /\.scss$/,
                use: [
                    'style-loader',
                    'css-loader',
                    {
                        loader: 'sass-loader', // compiles Sass to CSS
                        options: {
                            includePaths : [nodeModulesDir]
                        }
                    }
                ]
            },
            {
                test: /\.(ttf|eot|woff|woff2)$/,
                use: {
                    loader: 'file-loader',
                    options: {
                        name: 'fonts/[name].[ext]',
                    },
                },
            }
        ]
    },
    plugins: [
        new CheckerPlugin()
    ],
    resolve: {
        alias: {
            'react': 'preact-compat',
            'react-dom': 'preact-compat'
        },
        modules: [
            nodeModulesDir,
            srcDir
        ],
```

```
        extensions: ['.js', '.json', '.ts', '.tsx'],
        mainFields: ['module', 'main'],
    },
};
```

B.2.2 Pasta archetypes

Arquivo default.md

```
---
title: "{{ replace .Name "-" " " | title }}"
date: {{ .Date }}
draft: true
---
```

B.2.3 Pasta content

Arquivo cadastrar-plano.md

```
---
title: Cadastrar Plano
url: /planos/cadastrar/
---
```

Arquivo cadastrar-universidade.md

```
---
title: Cadastrar Universidade
url: /universidades/cadastrar/
---
```

Arquivo cadastro.md

```
---
title: Cadastro
url: /cadastro
---
```

Arquivo conta.md

```
---
title: Conta
url: /conta/
---
```

Arquivo detalhes-disciplina.md

```
---
title: Detalhes da Disciplina
url: /disciplinas/detalhes/
---
```

Arquivo detalhes-oferta-disciplina.md

```
---
title: Detalhes da Oferta de Disciplina
url: /ofertas-disciplinas/detalhes/
---
```

Arquivo detalhes-turma.md

```
---
title: Detalhes da Turma
url: /turmas/detalhes/
---
```

Arquivo editar-plano.md

```
---
title: Editar Plano
url: /planos/editar/
---
```

Arquivo editar-universidade.md

```
---
title: Editar Universidade
url: /universidades/editar/
---
```

Arquivo entrar.md

```
---
title: Entrar
url: /entrar
---
```

Arquivo planos.md

```
---
title: Planos
url: /planos/
---
```

Arquivo recuperar-senha.md

```
---
title: Recuperar Senha
url: /recuperar-senha
---
```

Arquivo sobre.md

```
---
title: Sobre
author: Fernando Jorge Mota
---
```

Arquivo universidades.md

```
---
title: Universidades
url: /universidades
---
```

B.2.4 Pasta layouts

Arquivo index.html

```
{{ partial "page.html" . }}
```

Arquivo single.html

```
{{ partial "page.html" . }}
```

B.2.5 Pasta static

Arquivo manifest.webmanifest

```
{
  "name": "Guru da Matrícula",
  "short_name": "GDM",
  "theme_color": "#ff5722",
  "background_color": "#29b6f6",
  "display": "standalone",
  "Scope": "/",
  "start_url": "/",
  "icons": null,
  "splash_pages": null
}
```

Arquivo robots.txt

```
User-agent: *
Allow: /
```

B.2.6 Pasta layouts/_default

Arquivo 404.html

```
{{ partial "page.html" . }}
```

Arquivo list.html

```
{{ partial "page.html" . }}
```

Arquivo list.json.json

```
{
    "pages": {
        "actual": {{ .Paginator.PageNumber | jsonify }},
```

```
        "total_elements": {{ .Paginator.TotalNumberOfElements |
        ↪ jsonify}},
        "total_pages": {{ .Paginator.TotalPages | jsonify}},
        "prev": {{ if .Paginator.HasPrev }}{{ .Paginator.Prev.URL |
        ↪ jsonify }}{{ else }}null{{ end }},
        "next": {{ if .Paginator.HasNext }}{{ .Paginator.Next.URL |
        ↪ jsonify }}{{ else }}null{{ end }},
        "first": {{ .Paginator.First.URL | jsonify}},
        "last": {{ .Paginator.Last.URL | jsonify }}
    },
    "items": [
        {{ range $index, $page := .Pages }}
            {{ if gt $index 0 }}, {{ end }}
            {
                "title": {{ $page.Title | jsonify }},
                "summary": {{ $page.Summary | jsonify }},
                "truncated": {{ $page.Truncated | jsonify }},
                "url": {{ .RelPermalink | jsonify }},
                "date": {{ .Date | jsonify}}
            }
        {{ end }}
    ]
}
```

Arquivo single.html

```
{{ partial "page.html" . }}
```

Arquivo single.json.json

```
{
    "title": {{ .Title | jsonify }},
    "content": {{ .Content | jsonify }},
    "date": {{ .Date | jsonify }},
    "related": [
        {{ $related := (where (.Site.RegularPages.Related .) "Type"
        ↪ .Type) | first 5 }}
        {{ with $related }}
```

```
            {{ range $index, $page := . }}
                {{ if gt $index 0 }},{{ end }}
                {
                    "url": {{ $page.RelPermalink | jsonify }},
                    "title": {{ $page.Title | jsonify }}
                }
            {{ end }}
        {{ end }}
    ]
}
```

B.2.7 Pasta layouts/partials

Arquivo link.html

```
{{if eq (substr . (sub (len .) 4)) ".css"}}
    <link rel="stylesheet" type="text/css" href="/static/{{ . }}" />
{{ end }}
```

Arquivo page.html

```
{{ $data := getJSON "static/static/stats.json" }}
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <title>Guru da Matrícula</title>
        <meta name="viewport" content="width=device-width,
        ↪  initial-scale=1, maximum-scale=5, minimum-scale=1,
        ↪  user-scalable=yes, minimal-ui, viewport-fit=cover" />
        <meta name="apple-mobile-web-app-capable" content="yes" />
        <meta charset="utf-8" />
        <meta name="theme-color" content="#b72114" />
        <link rel="manifest" href="/manifest.webmanifest">
        <!-- TODO: need to find a way to load css dynamically -->
        {{ range $bundle := $data.assetsByChunkName }}
            {{ if reflect.IsSlice $bundle }}
                {{ range $element := $bundle }}
                    {{ partial "link.html" $element}}
                {{ end }}
            {{ else }}
```

```
                {{ partial "link.html" $bundle}}
            {{ end }}
        {{ end }}
    </head>
    <body>
        <div id="container"></div>
        {{ range $bundle := $data.assetsByChunkName }}
            {{ if reflect.IsSlice $bundle }}
                {{ range $element := $bundle }}
                    {{ partial "script.html" $element }}
                {{ end }}
            {{ else }}
                {{ partial "script.html" $bundle }}
            {{ end }}
        {{ end }}
    </body>
</html>
```

Arquivo script.html

```
{{if eq (substr . (sub (len .) 3)) ".js"}}
    <script language="javascript" type="text/javascript" src="/static/{{ .
    ↪ }}"></script>
{{ end }}
```

B.2.8 Pasta src/css

Arquivo app.scss

```
// Source:
↪ https://material.io/resources/color/#!/?view.left=1&view.right=0&primary.color=
$mdc-theme-primary: #ff5722;
$mdc-theme-secondary: #29b6f6;

@import "~normalize.css/normalize.css";
@import "~material-design-icons/iconfont/material-icons.css";
@import "~@rmwc/data-table/data-table.css";
@import "~@rmwc/icon/icon.css";

@import "@material/card/mdc-card.scss";
```

```
11 @import "@material/chips/mdc-chips.scss";
12 @import "@material/drawer/mdc-drawer.scss";
13 @import "@material/dialog/mdc-dialog.scss";
14 @import "@material/icon-button/mdc-icon-button.scss";
15 @import "@material/linear-progress/mdc-linear-progress.scss";
16 @import "@material/list/mdc-list.scss";
17 @import "@material/layout-grid/mdc-layout-grid.scss";
18 @import "@material/button/mdc-button.scss";
19 @import "@material/select/mdc-select.scss";
20 @import "@material/floating-label/mdc-floating-label.scss";
21 @import "@material/notched-outline/mdc-notched-outline.scss";
22 @import "@material/line-ripple/mdc-line-ripple.scss";
23 @import "@material/list/mdc-list.scss";
24 @import "@material/menu/mdc-menu.scss";
25 @import "@material/menu-surface/mdc-menu-surface.scss";
26 @import "@material/snackbar/mdc-snackbar.scss";
27 @import "@material/tab/mdc-tab.scss";
28 @import "@material/tab-indicator/mdc-tab-indicator.scss";
29 @import "@material/tab-bar/mdc-tab-bar.scss";
30 @import "@material/tab-scroller/mdc-tab-scroller.scss";
31 @import "@material/top-app-bar/mdc-top-app-bar.scss";
32 @import "@material/switch/mdc-switch.scss";
33 @import "@material/textfield/mdc-text-field.scss";
34 @import "@material/typography/mdc-typography.scss";
35
36 @import "@material/elevation/mdc-elevation.scss";
37 @import "@material/ripple/mdc-ripple";
38 @import "@material/shape/mixins";
39 @import "@material/shape/functions";
40
41 .results .result {
42   padding: 16px;
43 }
44
45 /* roboto-300 - latin */
46 @font-face {
47   font-family: "Roboto";
48   font-style: normal;
49   font-weight: 300;
```

```
50    src: local("Roboto Light"), local("Roboto-Light"),
51      url("../fonts/roboto-v18-latin-300.woff2") format("woff2"),
52      /* Chrome 26+, Opera 23+, Firefox 39+ */ ma
53        url("../fonts/roboto-v18-latin-300.woff") format("woff"); /* Chrome
          ↪  6+, Firefox 3.6+, IE 9+, Safari 5.1+ */
54  }
55
56  /* roboto-regular - latin */
57  @font-face {
58    font-family: "Roboto";
59    font-style: normal;
60    font-weight: 400;
61    src: local("Roboto"), local("Roboto-Regular"),
62      url("../fonts/roboto-v18-latin-regular.woff2") format("woff2"),
63      /* Chrome 26+, Opera 23+, Firefox 39+ */
64        url("../fonts/roboto-v18-latin-regular.woff") format("woff"); /*
          ↪  Chrome 6+, Firefox 3.6+, IE 9+, Safari 5.1+ */
65  }
66  /* roboto-500 - latin */
67  @font-face {
68    font-family: "Roboto";
69    font-style: normal;
70    font-weight: 500;
71    src: local("Roboto Medium"), local("Roboto-Medium"),
72      url("../fonts/roboto-v18-latin-500.woff2") format("woff2"),
73      /* Chrome 26+, Opera 23+, Firefox 39+ */
74        url("../fonts/roboto-v18-latin-500.woff") format("woff"); /* Chrome
          ↪  6+, Firefox 3.6+, IE 9+, Safari 5.1+ */
75  }
76
77  body {
78    font-family: "Roboto";
79    overscroll-behavior: contain;
80  }
81
82  .top-app {
83    opacity: 1;
84    visibility: visible;
85    transition: visibility 0s, opacity 0.2s ease 0.2s;
```

```
86    &.hide {
87      visibility: hidden;
88      opacity: 0;
89      transition: opacity 0.2s, visibility 0s ease 0.2s;
90    }
91  }
92
93  .top-app-search {
94    position: fixed;
95    display: block;
96    background: #fff;
97    width: 100%;
98    top: 0;
99    z-index: 10;
100   margin-bottom: -64px;
101   .mdc-text-field {
102     height: 64px;
103     vertical-align: middle;
104   }
105   .mdc-text-field__input {
106     padding-left: 80px !important;
107   }
108   .mdc-text-field__icon {
109     margin-right: $mdc-top-app-bar-section-horizontal-padding;
110     margin-bottom: $mdc-top-app-bar-section-vertical-padding;
111     &:focus {
112       outline: 0;
113     }
114     @include mdc-ripple-surface();
115     @include mdc-ripple-radius-bounded;
116     @include mdc-states;
117   }
118 }
119
120 @import "./better-calendar.scss";
121
122 .form-row {
123   margin-bottom: 15px;
124   a {
```

```
125     margin-left: 10px;
126   }
127 }
```

Arquivo better-calendar.scss

```
1  .better-calendar {
2    touch-action: none;
3    #header,
4    #content {
5      width: 100%;
6      height: 100%;
7      display: grid;
8      grid-gap: 0px;
9      grid-template-columns: 7ch calc(100% - 7ch);
10   }
11   .column {
12     touch-action: none;
13   }
14   #header {
15     overflow: hidden;
16     height: 30px;
17     .cell {
18       height: 30px;
19     }
20   }
21   #content {
22     overflow-y: hidden;
23     height: calc(100% - 30px);
24     overflow-x: hidden;
25   }
26   .column-data {
27     position: relative;
28     width: 100%;
29     height: calc(24 * var(--hourHeight));
30     .overlay,
31     .data {
32       width: 100%;
33       height: 100%;
```

```
      position: absolute;
      top: 0;
      left: 0;
    }
    .overlay {
      z-index: 10;
    }
  }

  .item {
    border-top: 2px #ccc solid;
  }
  .inner,
  .hours {
    height: calc(var(--visibleHours, 20) * var(--hourHeight));
  }
  .inner,
  .weekdays {
    display: grid;
    overflow: hidden;
    grid-template-columns: repeat(
      var(--totalColumns),
      calc(100% / var(--visibleColumns))
    );
  }
  .inner .column {
    transform: translate(
      calc(var(--column, 0) * -100% + var(--columnPosition, 0px)),
      calc(var(--row, 0) * var(--hourHeight) * -1 + var(--rowPosition,
      ↪ 0px))
    );
  }
  .weekdays .cell {
    transform: translate(
      calc(var(--column, 0) * -100% + var(--columnPosition, 0px))
    );
  }
  .hours {
    overflow: hidden;
```

```
72      .cell {
73        transform: translate(
74          0,
75          calc(var(--row, 0) * var(--hourHeight) * -1 + var(--rowPosition,
            ↪  0px))
76        );
77      }
78    }
79    .cell {
80      height: var(--hourHeight);
81      border-left: 1px #ccc solid;
82      box-sizing: border-box;
83      overflow: hidden;
84      text-overflow: ellipsis;
85    }
86    .event {
87      display: grid;
88      grid-gap: 0px;
89      @include mdc-shape-radius($mdc-shape-medium-component-radius);
90      @include mdc-elevation(5);
91      font-size: 12px;
92      text-overflow: ellipsis;
93      .time {
94        text-align: left;
95      }
96      .title {
97        vertical-align: middle;
98      }
99    }
100   .event-small {
101     grid-template-columns: 11ch 1fr;
102     .title {
103       vertical-align: top;
104     }
105   }
106   @media (min-width: 480px) {
107     .event-small .title::before {
108       content: " - ";
109     }
```

```
110   }
111   @media (max-width: 480px) {
112     .weekdays .cell {
113       text-align: center;
114     }
115     .event {
116       overflow: hidden;
117       .time {
118         display: none;
119       }
120     }
121   }
122   .placeholder {
123     background: transparent;
124     &.hover {
125       @include mdc-shape-radius($mdc-shape-medium-component-radius);
126       background: var(--hoverColor);
127     }
128   }
129   &.animate {
130     .hours .cell,
131     .inner .column,
132     .weekdays .cell {
133       transition: transform calc(var(--factor) * 0.5s)
134         cubic-bezier(1, 1.59, 0.61, 0.74) 0.1s;
135     }
136   }
137 }
```

B.2.9 Pasta src/js

Arquivo app.tsx

```
1 // import "preact/debug";
2 import React from "react";
3 import ReactDOM from "react-dom";
4 import { Provider } from "react-redux";
5 import { getSearch, ConnectedRouter } from "connected-react-router";
6 import "../css/app.scss";
7 import pages from "./pages";
```

```
8  import getAuthModule from "./pages/base/store/auth";
9  import Title from "./pages/base/components/title";
10 import { APP_NAME } from "./config";
11 import client from "./client";
12 import { ApolloProvider } from "@apollo/react-hooks";
13 import store from "./store";
14 import historyInstance from "./history";
15 import { DynamicModuleLoader } from "redux-dynamic-modules";
16 import { Workbox } from "workbox-window";
17
18 const container = document.querySelector("#container");
19
20 function init() {
21   if (!container) {
22     alert("Erro ao renderizar conteúdo. Container não encontrado");
23     return;
24   }
25   ReactDOM.render(
26     <Provider store={store}>
27       <ApolloProvider client={client}>
28         <DynamicModuleLoader modules={getAuthModule()}>
29           <Title title={APP_NAME}>
30             <ConnectedRouter
               ↪  history={historyInstance}>{pages}</ConnectedRouter>
31           </Title>
32         </DynamicModuleLoader>
33       </ApolloProvider>
34     </Provider>,
35     container
36   );
37 }
38 (window as any).getSearch = getSearch;
39 (window as any).store = store;
40 if (module.hot && process.env.NODE_ENV !== "production") {
41   console.log("Hot-loading enabled");
42   module.hot.accept("./pages", init);
43 }
44 if ("serviceWorker" in navigator) {
45   // const wb = new Workbox("/sw.js");
```

```
46    // wb.register();
47  }
48
49  init();
```

Arquivo client.ts

```
1   import { BASE_BACKEND_URL } from "./config";
2   import { ApolloClient } from "apollo-client";
3   import { InMemoryCache } from "apollo-cache-inmemory";
4   import { HttpLink } from "apollo-link-http";
5   import { onError } from "apollo-link-error";
6   import { ApolloLink } from "apollo-link";
7   import { setContext } from "apollo-link-context";
8   import store from "./store";
9   import { getCSRFToken } from "./pages/base/store/csrf/selectors";
10
11  function getNewContext(context: any, token: string) {
12    return {
13      ...context,
14      headers: {
15        ...context.headers,
16        "X-CSRF-Token": token
17      }
18    };
19  }
20
21  const csrf = setContext((request, context) => {
22    // add the authorization to the headers
23    let token = getCSRFToken(store.getState());
24
25    if (!token) {
26      return new Promise(resolve => {
27        let unsubscribe = store.subscribe(() => {
28          token = getCSRFToken(store.getState());
29          if (!!token) {
30            unsubscribe();
31            resolve(getNewContext(context, token));
32          }
33        });
```

```
    });
  }
  return getNewContext(context, token);
});

const error = onError(({ graphQLErrors, networkError }) => {
  if (graphQLErrors)
    graphQLErrors.forEach(({ message, locations, path }) =>
      console.log(
        `[GraphQL error]: Message: ${message}, Location: ${locations},
        ↪  Path: ${path}`
      )
    );
  if (networkError) console.log(`[Network error]: ${networkError}`);
});

const http = new HttpLink({
  uri: `${BASE_BACKEND_URL}/api/graphql`,
  credentials: "include"
});

export default new ApolloClient({
  link: ApolloLink.from([error, csrf, http]),
  cache: new InMemoryCache()
});
```

Arquivo custom.d.ts

```
declare module "worker-loader!*" {
  class WebpackWorker extends Worker {
    constructor();
  }

  export default WebpackWorker;
}

interface System {
  import(request: string): Promise<any>;
}

```

```
13 declare var System: System;
14
15 declare module "*.graphql" {
16   const content: string;
17   export default content;
18 }
```

Arquivo history.ts

```
1 import { createBrowserHistory } from "history";
2 export default createBrowserHistory();
```

Arquivo schema.graphql

```
1  schema {
2    query: Query
3    mutation: Mutation
4  }
5
6  ################### Mutation/Inputs ###################
7  input PlanHourMinuteInput {
8    hour: Int!
9    minute: Int!
10 }
11
12 input PlanRoomInput {
13   id: ID!
14   genericID: String!
15   name: String!
16   olcode: String
17   description: String
18 }
19
20 input PlanScheduleInput {
21   start: PlanHourMinuteInput!
22   end: PlanHourMinuteInput!
23   dayOfWeek: Int!
24   room: PlanRoomInput
25 }
```

```
26
27 input PlanTeacherInput {
28   id: ID!
29   genericID: String!
30   name: String!
31   url: String
32 }
33
34 input PlanCampusInput {
35   id: ID!
36   name: String!
37 }
38
39 input PlanVacancyInput {
40   offered: Int!
41   filled: Int!
42 }
43
44 input PlanTableInput {
45   id: ID!
46   title: String!
47   description: String
48   columns: [PlanTableColumnInput!]!
49   rows: [PlanTableRowInput!]!
50 }
51
52 input PlanTableColumnInput {
53   id: ID!
54   name: String!
55 }
56
57 input PlanTableRowInput {
58   row: Int!
59   column: Int!
60   value: String!
61 }
62
63 input PlanCourseOfferInput {
64   id: ID!
```

```
65    name: String!
66    exclusivity: Exclusivity! = Unknown
67  }
68
69  input PlanTeamInput {
70    id: ID!
71    code: String!
72    version: String!
73    schedules: [PlanScheduleInput!]!
74    teachers: [PlanTeacherInput!]!
75    vacancies: PlanVacancyInput
76    course: PlanCourseOfferInput
77    tables: [PlanTableInput!]!
78  }
79
80  input PlanDisciplineOfferInput {
81    id: ID!
82    genericID: ID
83    campus: ForeignKeyInput
84    version: String!
85    custom: Boolean!
86    code: String!
87    name: String!
88    teams: [PlanTeamInput!]!
89  }
90
91  input PlanDisciplineRelatedItemInput {
92    type: String!
93    value: String!
94    description: String
95  }
96
97  input PlanDisciplineRelatedInput {
98    name: String!
99    type: String!
100   items: [PlanDisciplineRelatedItemInput!]!
101 }
102
103 input PlanDisciplineRelatedDisciplineSummary {
```

```
  id: ID!
  genericID: ID!
  course: ForeignKeyInput!
  habilitation: ForeignKeyInput!
  step: ForeignKeyInput!
  version: String!
  code: String!
  name: String!
  type: String!
  description: String!
}

input PlanDisciplineInput {
  id: ID!
  genericID: ID!
  course: ForeignKeyInput!
  habilitation: ForeignKeyInput!
  step: ForeignKeyInput!
  version: String!
  code: String!
  name: String!
  type: String!
  description: String!
  related: [PlanDisciplineRelatedInput!]!
  relatedDisciplines: [PlanDisciplineRelatedDisciplineSummary!]!
  tables: [PlanTableInput!]!
}

input PlanTeamSelectionInput {
  offerID: ID!
  offerVersion: String!
  teamID: ID!
  color: String!
  enabled: Boolean!
  selected: Boolean!
}

input PlanDisciplineSelectionInput {
  disciplineID: ID!
```

```
143   disciplineVersion: String!
144   enabled: Boolean!
145 }
146
147 input PlanPossibilityInput {
148   name: String!
149   selectionTeams: [PlanTeamSelectionInput!]!
150   selectionDisciplines: [PlanDisciplineSelectionInput!]!
151 }
152
153 input ForeignKeyInput {
154   id: ID!
155   name: String!
156 }
157
158 input PlanInput {
159   id: ID
160   name: String!
161   public: Boolean!
162   universityID: ID!
163   periodID: ID!
164   disciplines: [PlanDisciplineInput!]!
165   offers: [PlanDisciplineOfferInput!]!
166   possibilities: [PlanPossibilityInput!]!
167   selectedPossibility: Int!
168 }
169
170 input UserPasswordInput {
171   oldPassword: String!
172   password: String!
173   confirmPassword: String!
174 }
175
176 input UserDisciplineInput {
177   id: ID!
178   genericID: ID!
179   code: String!
180   name: String!
181 }
```

```
182
183 input UserInput {
184   name: String!
185   password: UserPasswordInput
186   university: ForeignKeyInput
187   courses: [ForeignKeyInput!]
188   habilitations: [ForeignKeyInput!]
189   completedDisciplines: [UserDisciplineInput!]!
190 }
191
192 input UniversityDataInput {
193   baseUrl: String!
194   deleteUrl: String!
195   aboutUrl: String!
196   generateSecret: Boolean!
197 }
198
199 input UniversityInput {
200   id: ID
201   acronym: String!
202   name: String!
203   public: Boolean!
204   habilitations: UniversityDataInput!
205   offers: UniversityDataInput!
206 }
207
208 type Mutation {
209   setUniversity(university: UniversityInput!): University!
210   setPlan(plan: PlanInput!): Plan
211   deletePlan(id: String!): Boolean!
212   setUser(user: UserInput!): User!
213   deleteUser(password: String!): Boolean!
214 }
215
216 ################### Query/Types ###################
217
218 ################ Search ################
219
220 enum SearchOperator {
```

```
221   All
222   And
223   Or
224   Not
225   Terminal
226   FieldTerminal
227 }
228
229 input SearchExpression {
230   type: SearchOperator!
231   expressions: [SearchExpression!]
232   attribute: String
233   value: String
234 }
235
236 enum SearchFilter {
237   NONE
238   PREREQUISITES
239   COURSES
240   ALL
241 }
242
243 input SearchOptions {
244   version: String
245   before: Int
246   after: Int
247   limit: Int
248   filter: SearchFilter = NONE
249 }
250
251 type SearchQueryInfo {
252   version: String!
253   last: Int!
254   first: Int!
255   count: Int!
256 }
257
258 type SearchPageInfo {
259   after: Int!
```

```
260   before: Int!
261 }
262
263 type SearchResultInfo {
264   query: SearchQueryInfo!
265   page: SearchPageInfo!
266 }
267
268 interface SearchResult {
269   info: SearchResultInfo!
270 }
271
272 ################ Plans ################
273
274 type PlanTeamSelection {
275   offerID: ID!
276   offerVersion: String!
277   teamID: ID!
278   color: String!
279   selected: Boolean!
280   enabled: Boolean!
281 }
282
283 type PlanDisciplineSelection {
284   disciplineID: ID!
285   disciplineVersion: String!
286   enabled: Boolean!
287 }
288
289 type PlanPossibility {
290   name: String!
291   disciplines: [PlanDisciplineSelection!]!
292   teams: [PlanTeamSelection!]!
293 }
294
295 scalar Time
296
297 type PlanDisciplineOfferData {
298   id: ID!
```

```
  genericID: ID!
  university: University!
  campus: Campus!
  period: Period!
  version: String!
  code: String!
  name: String!
  teams: [Team!]!
  team(id: ID!): Team
}

type PlanDisciplineOffer {
  custom: Boolean!
  deleted: Boolean!
  updated: DisciplineOffer
  offer: PlanDisciplineOfferData!
}

type PlanDisciplineSummary {
  id: ID!
  genericID: ID!
  course: Course!
  habilitation: Habilitation!
  step: Step!
  version: String!
  code: String!
  name: String!
  type: String!
  description: String!
}

type PlanDisciplineData {
  id: ID!
  version: String!
  university: University!
  course: Course!
  habilitation: Habilitation!
  step: Step!
  genericID: ID!
```

```
  code: String!
  name: String!
  description: String!
  type: String!
  tables: [Table!]!
  related: [DisciplineRelated!]!
  relatedDisciplines: [PlanDisciplineSummary!]!
}

type PlanDiscipline {
  deleted: Boolean!
  updated: Discipline
  discipline: PlanDisciplineData!
}

type PlanSelectedSchedule {
  offerID: ID!
  title: String!
  color: String!
  start: HourMinute!
  end: HourMinute!
  dayOfWeek: Int!
}

type PlanVersion {
  id: ID!
  savedAt: Time!
  selectedPossibility: Int!
  disciplines: [PlanDiscipline!]!
  disciplineOffers: [PlanDisciplineOffer!]!
  possibilities: [PlanPossibility!]!
  totalPossibilities: Int!
  selectedSchedules: [PlanSelectedSchedule!]!
}

type Plan {
  id: ID!
  name: String!
  lastModified: Time!
```

```
377   createdAt: Time!
378   user: User!
379   public: Boolean!
380   university: University!
381   period: Period!
382   versions(start: Int, end: Int): [PlanVersion!]!
383   versionByIndex(index: Int!): PlanVersion
384   versionByID(id: ID!): PlanVersion
385 }
386
387 ################ User ################
388
389 type UserDiscipline {
390   id: ID!
391   genericID: ID!
392   code: String!
393   name: String!
394 }
395
396 type User {
397   id: String!
398   name: String!
399   email: String!
400   oauth2_provider: String!
401   plans: [Plan!]!
402   university: University
403   courses: [Course!]
404   habilitations: [Habilitation!]
405   completedDisciplines: [UserDiscipline!]!
406 }
407
408 ################ Tables ################
409
410 type TableColumn {
411   id: ID!
412   name: String!
413 }
414
415 type TableRow {
```

```
  row: Int!
  column: Int!
  value: String!
}

type Table {
  id: ID!
  title: String!
  description: String
  columns: [TableColumn!]!
  rows: [TableRow!]!
}

################ Offers ################

enum Exclusivity {
  Exclusive
  NotExclusive
  Unknown
}

type CourseOffer {
  id: ID!
  name: String!
  exclusivity: Exclusivity!
}

type Vacancy {
  offered: Int!
  filled: Int!
}

type Teacher {
  id: ID!
  genericID: String!
  university: University!
  period: Period!
  name: String!
  url: String
```

```
455   searchTeams(
456     periodID: ID
457     query: SearchExpression
458     options: SearchOptions
459   ): TeamSearchResult!
460 }
461
462 type TeacherSearchResult implements SearchResult {
463   info: SearchResultInfo!
464   results: [Teacher!]!
465 }
466
467 type HourMinute {
468   hour: Int!
469   minute: Int!
470 }
471
472 type RoomSearchResult implements SearchResult {
473   info: SearchResultInfo!
474   results: [Room!]!
475 }
476
477 type Room {
478   id: ID!
479   genericID: String!
480   university: University!
481   campus: Campus!
482   period: Period!
483   name: String!
484   olcode: String
485   description: String
486   searchTeams(
487     periodID: ID
488     query: SearchExpression
489     options: SearchOptions
490   ): TeamSearchResult!
491 }
492
493 type Schedule {
```

```
494   start: HourMinute!
495   end: HourMinute!
496   dayOfWeek: Int!
497   room: Room
498 }
499
500 type DisciplineOfferSearchResult implements SearchResult {
501   info: SearchResultInfo!
502   results: [DisciplineOffer!]!
503 }
504
505 type DisciplineOffer {
506   id: ID!
507   genericID: ID!
508   university: University!
509   campus: Campus!
510   period: Period!
511   version: String!
512   code: String!
513   name: String!
514   teams: [Team!]!
515   team(id: ID!): Team
516   searchTeams(
517     query: SearchExpression
518     options: SearchOptions
519   ): TeamSearchResult!
520 }
521
522 type TeamSearchResult implements SearchResult {
523   info: SearchResultInfo!
524   results: [Team!]!
525 }
526
527 type Team {
528   id: ID!
529   code: String!
530   version: String!
531   schedules: [Schedule!]!
532   teachers: [Teacher!]!
```

```
533   vacancies: Vacancy
534   university: University!
535   campus: Campus!
536   period: Period!
537   discipline: DisciplineOffer!
538   course: CourseOffer
539   tables: [Table!]!
540 }
541
542 type Campus {
543   id: ID!
544   university: University!
545   version: String!
546   period: Period!
547   name: String!
548   disciplineOffers: [DisciplineOffer!]!
549   searchDisciplineOffers(
550     query: SearchExpression
551     options: SearchOptions
552   ): DisciplineOfferSearchResult!
553   searchTeams(
554     query: SearchExpression
555     options: SearchOptions
556   ): TeamSearchResult!
557 }
558
559 type Period {
560   id: ID!
561   university: University!
562   version: String!
563   name: String!
564   campi: [Campus!]!
565 }
566
567 ################ Semantic ################
568
569 enum DisciplineRelatedItemType {
570   DISCIPLINE_ID
571   DISCIPLINE_GENERIC_ID
```

```
572   TEXT
573 }
574
575 type DisciplineRelatedItem {
576   type: DisciplineRelatedItemType!
577   value: String!
578   description: String
579 }
580
581 enum DisciplineRelatedType {
582   PREREQUISITE
583   SET
584   EQUIVALENT
585 }
586
587 type DisciplineRelated {
588   name: String!
589   type: DisciplineRelatedType!
590   items: [DisciplineRelatedItem!]!
591 }
592
593 type DisciplineSearchResult implements SearchResult {
594   info: SearchResultInfo!
595   results: [Discipline!]!
596 }
597
598 type Discipline {
599   id: ID!
600   version: String!
601   university: University!
602   course: Course!
603   habilitation: Habilitation!
604   step: Step!
605   genericID: ID!
606   code: String!
607   name: String!
608   description: String!
609   type: String!
610   tables: [Table!]!
```

```
611   searchTeams(
612     periodID: ID!
613     query: SearchExpression
614     options: SearchOptions
615   ): TeamSearchResult!
616   searchDisciplineOffers(
617     periodID: ID!
618     query: SearchExpression
619     options: SearchOptions
620   ): DisciplineOfferSearchResult!
621   related: [DisciplineRelated!]!
622   relatedDisciplines: [Discipline!]!
623 }
624
625 type HabilitationLimitValue {
626   id: Int!
627   key: String!
628   value: Float!
629 }
630
631 type HabilitationLimit {
632   id: ID!
633   name: String!
634   context: String!
635   unit: String!
636   values: [HabilitationLimitValue!]!
637 }
638
639 type Step {
640   id: ID!
641   name: String!
642   disciplines: [Discipline!]!
643 }
644
645 type Habilitation {
646   id: ID!
647   university: University!
648   course: Course!
649   version: String!
```

```
650   name: String!
651   steps: [Step!]!
652   tables: [Table!]!
653   limits: [HabilitationLimit!]!
654 }
655
656 type Course {
657   id: ID!
658   university: University!
659   name: String!
660   version: String!
661   courseVersion: String!
662   url: String
663   habilitations: [Habilitation!]!
664 }
665
666 type UniversityMutex {
667   lastUpdated: Time!
668 }
669
670 type UniversityData {
671   baseUrl: String!
672   deleteUrl: String!
673   aboutUrl: String!
674   lastUpdated: Time
675   queueSize: Int!
676   secret: String!
677 }
678
679 type University {
680   id: ID!
681   acronym: String!
682   name: String!
683   public: Boolean!
684   periods: [Period!]!
685   courses: [Course!]!
686   offers: UniversityData!
687   habilitations: UniversityData!
688 }
```

```
689
690 ################### Offline System ###################
691
692 input OfflineIDVersion {
693   ID: String!
694   Version: String!
695 }
696
697 input OfflineRequest {
698   universityID: String!
699   periodID: String!
700   courseID: String!
701   habilitations: [OfflineIDVersion!]!
702   disciplineOffers: [OfflineIDVersion!]!
703   disciplines: [OfflineIDVersion!]!
704 }
705
706 type OfflineResponse {
707   disciplineOffers: [DisciplineOffer!]!
708   disciplineOffersToDelete: [String!]!
709   disciplines: [Discipline!]!
710   disciplinesToDelete: [String!]!
711   habilitations: [Habilitation!]!
712   habilitationsToDelete: [String!]!
713 }
714
715 type Query {
716   university(id: ID!): University
717   universities(onlyOwned: Boolean = false): [University!]!
718   course(id: ID!): Course
719   coursesByUniversity(universityID: ID!): [Course!]!
720   coursesByIDs(ids: [ID!]!): [Course!]!
721   period(id: ID!): Period
722   periodsByUniversity(universityID: ID!): [Period!]!
723   periodsByIDs(ids: [ID!]!): [Period!]!
724   campus(id: ID!): Campus
725   campiByPeriod(periodID: ID!): [Campus!]!
726   campiByIDs(ids: [ID!]!): [Campus!]!
727   habilitation(id: ID!): Habilitation
```

```
  habilitationsByCourse(courseID: ID!): [Habilitation!]!
  habilitationsByCourses(coursesIDs: [ID!]!): [Habilitation!]!
  habilitationsByIDs(ids: [ID!]!): [Habilitation!]!
  disciplineOffer(id: ID!): DisciplineOffer
  disciplineOffersByCampus(campusID: ID!): [DisciplineOffer!]!
  disciplineOffersByIDs(ids: [ID!]!): [DisciplineOffer!]!
  discipline(id: ID!): Discipline
  disciplinesByHabilitationAndStep(
    habilitationID: ID!
    stepID: ID!
  ): [Discipline!]!
  disciplinesByIDs(ids: [ID!]!): [Discipline!]!
  team(disciplineID: ID!, teamID: ID!): Team
  teams(disciplineID: ID!): [Team!]!
  loggedUser: User
  getOfflineUpdates(request: [OfflineRequest!]!): OfflineResponse!
  searchTeachers(
    periodID: ID!
    query: SearchExpression
    options: SearchOptions
  ): TeacherSearchResult!
  searchDisciplines(
    universityID: ID!
    query: SearchExpression
    options: SearchOptions
  ): DisciplineSearchResult!
  searchDisciplineOffers(
    periodID: ID!
    query: SearchExpression
    options: SearchOptions
  ): DisciplineOfferSearchResult!
  searchTeams(
    periodID: ID!
    query: SearchExpression
    options: SearchOptions
  ): TeamSearchResult!
  plan(id: ID!): Plan
  plans: [Plan!]!
}
```

Arquivo store.ts

```
import getUIModule from "./pages/base/store/ui";
import getCSRFModule from "./pages/base/store/csrf";
import {
  RouterState,
  connectRouter,
  routerMiddleware,
  RouterRootState
} from "connected-react-router";
import { createStore, IModule } from "redux-dynamic-modules-core";
import { getSagaExtension } from "redux-dynamic-modules-saga";
import { Reducer, AnyAction } from "redux";
import historyInstance from "./history";

const routerModule: IModule<RouterRootState> = {
  id: "router",
  reducerMap: {
    router: connectRouter(historyInstance) as Reducer<RouterState,
    ↪  AnyAction>
  },
  middlewares: [routerMiddleware(historyInstance)]
};

export default createStore(
  {
    extensions: [getSagaExtension()]
  },
  routerModule,
  getUIModule(),
  getCSRFModule()
);
```

B.2.10 Pasta src/js/config

Arquivo dev.ts

```
export const BASE_BACKEND_URL = "https://127.0.0.1:8080";
export const APP_NAME = "Guru da Matrícula";
```

B.2.11 Pasta src/js/pages

Arquivo index.tsx

```
import { default as React, Component, Suspense, lazy } from "react";
import { Switch, Route } from "react-router";
import Title from "./base/components/title";
import Teacher from "./teacher";

function Loading() {
  return (
    <Title title="Carregando...">
      <div>Carregando...</div>
    </Title>
  );
}

const CreateAccount = lazy(() => import("./signup"));

const Login = lazy(() => import("./login"));

const Home = lazy(() => import("./home"));

const PlanDetails = lazy(() => import("./plan-details/"));

const CreatePlan = lazy(() => import("./create-plan/"));

const PasswordRecovery = lazy(() => import("./password-recovery/"));

const PlanList = lazy(() => import("./plan-list/"));

const CreateUniversity = lazy(() => import("./create-university/"));

const EditUniversity = lazy(() => import("./edit-university/"));

const Universities = lazy(() => import("./universities/"));

const Discipline = lazy(() => import("./discipline/"));

const Help = lazy(() => import("./help/"));
```

```
37
38 const Account = lazy(() => import("./account"));
39
40 const DisciplineOffer = lazy(() => import("./discipline-offer/"));
41
42 const Team = lazy(() => import("./team/"));
43
44 const Offline = lazy(() => import("./offline/"));
45
46 export default (
47   <Suspense fallback={<Loading />}>
48     <Switch>
49       <Route path="/entrar/">
50         <Login />
51       </Route>
52       <Route path="/cadastro/">
53         <CreateAccount />
54       </Route>
55       <Route path="/ajuda/">
56         <Help />
57       </Route>
58       <Route path="/conta/">
59         <Account />
60       </Route>
61       <Route path="/recuperar-senha/">
62         <PasswordRecovery />
63       </Route>
64       <Route path="/planos/cadastrar/">
65         <CreatePlan />
66       </Route>
67       <Route path="/universidades/cadastrar/">
68         <CreateUniversity />
69       </Route>
70       <Route path="/universidades/editar/">
71         <EditUniversity />
72       </Route>
73       <Route path="/disciplinas/detalhes/">
74         <Discipline />
75       </Route>
```

```
76          <Route path="/ofertas-disciplinas/detalhes/">
77            <DisciplineOffer />
78          </Route>
79          <Route path="/professor/detalhes/">
80            <Teacher />
81          </Route>
82          <Route path="/turmas/detalhes/">
83            <Team />
84          </Route>
85          <Route path="/universidades/">
86            <Universities />
87          </Route>
88          <Route path="/offline/">
89            <Offline />
90          </Route>
91          <Route path="/planos/editar/">
92            <PlanDetails />
93          </Route>
94          <Route path="/planos/">
95            <PlanList />
96          </Route>
97          <Route path="/">
98            <Home />
99          </Route>
100       </Switch>
101     </Suspense>
102 );
```

B.2.12 Pasta src/js/service-worker

Arquivo database.ts

```
1  import {
2    DBSchema,
3    IDBPDatabase,
4    StoreNames,
5    StoreKey,
6    StoreValue,
7    openDB
```

```
8  } from "idb";
9  import {
10   RemoteDiscipline,
11   RemoteDisciplineOffer,
12   RemoteTeam,
13   RemotePlan,
14   RemotePlanVersion,
15   RemoteUniversity,
16   RemotePeriod,
17   RemoteCampus,
18   RemoteHabilitation,
19   RemoteCourse,
20   RemoteSchedule,
21   RemoteRoom,
22   RemoteStep
23 } from "../graphql";
24
25 // TODO: Alguns dos campos listados abaixo são, basicamente, ForeignKeys
   ↪  que DEVEM
26 // ser melhoradas..Precisamos de alguma forma de poder fazer isso
27
28 export type ForeignKey = {
29   id: string;
30   name: string;
31 };
32
33 type AddForeignKey<T, S extends keyof any, M extends keyof any = never> =
   ↪ {
34   schema: Omit<T, S | M> &
35     {
36       [P in S]: string;
37     } &
38     {
39       [P in M]: string[];
40     };
41   resolver: Omit<T, S | M | "universityID">;
42 };
43
44 type FixUniversity<T> = Omit<T, "university"> & {
```

```
  universityID: string;
};

export type DisciplineDefinition = AddForeignKey<
  Omit<
    FixUniversity<RemoteDiscipline>,
    "searchTeams" | "searchDisciplineOffers" | "relatedDisciplines"
  >,
  "habilitation" | "step" | "course"
>;

export type DisciplineOfferDefinition = AddForeignKey<
  Omit<FixUniversity<RemoteDisciplineOffer>, "searchTeams">,
  "campus" | "period",
  "teams"
>;

export type CourseDefinition = AddForeignKey<
  FixUniversity<RemoteCourse>,
  never,
  "habilitations"
>;

export type RoomDefinition = AddForeignKey<
  Omit<RemoteRoom, "searchTeams" | "university">,
  "campus" | "period"
>;

type LocalTeamDefinition = AddForeignKey<
  FixUniversity<RemoteTeam>,
  "campus" | "period" | "discipline" | "course",
  "tables"
>;

export type TeamDefinition = {
  schema: LocalTeamDefinition["schema"];
  resolver: Omit<LocalTeamDefinition["resolver"], "schedules" |
  ↪  "teachers">;
};
```

```
83
84  export type PlanDefinition = AddForeignKey<
85    FixUniversity<RemotePlan>,
86    "period" | "user" | "version",
87    "versions"
88  >;
89
90  export type PlanVersionDefinition = AddForeignKey<
91    RemotePlanVersion,
92    never,
93    never
94  >;
95
96  export type UniversityDefinition = AddForeignKey<
97    RemoteUniversity,
98    never,
99    "periods" | "courses"
100 >;
101
102 export type PeriodDefinition = AddForeignKey<
103   FixUniversity<RemotePeriod>,
104   never,
105   "campi"
106 >;
107
108 export type CampusDefinition = AddForeignKey<
109   Omit<FixUniversity<RemoteCampus>, "searchDisciplineOffers" |
      ↪  "searchTeams">,
110   "period",
111   "disciplineOffers"
112 >;
113
114 export type StepDefinition = AddForeignKey<RemoteStep, never,
    ↪  "disciplines">;
115
116 export type HabilitationDefinition = AddForeignKey<
117   FixUniversity<RemoteHabilitation>,
118   "course",
119   "steps"
```

```
>;

export interface DatabaseSchema extends DBSchema {
  option: {
    key: string;
    value: {
      id: string;
      value: string;
    };
  };
  discipline: {
    key: string;
    value: DisciplineDefinition["schema"];
  };
  discipline_offer: {
    key: string;
    value: DisciplineOfferDefinition["schema"];
  };
  team: {
    key: string;
    value: TeamDefinition["schema"];
  };
  plan: {
    key: string;
    value: PlanDefinition["schema"];
  };
  plan_version: {
    key: string;
    value: PlanVersionDefinition["schema"];
  };
  university: {
    key: string;
    value: UniversityDefinition["schema"];
  };
  period: {
    key: string;
    value: PeriodDefinition["schema"];
  };
  campus: {
```

```
    key: string;
    value: CampusDefinition["schema"];
  };
  habilitation: {
    key: string;
    value: HabilitationDefinition["schema"];
  };
  course: {
    key: string;
    value: CourseDefinition["schema"];
  };
  step: {
    key: string;
    value: StepDefinition["schema"];
  };
  index: {
    key: string;
    value: {
      id: string;
      value: string;
    };
  };
}

export class Database<S = DatabaseSchema> {
  private db: IDBPDatabase<S>;
  constructor(db: IDBPDatabase<S>) {
    this.db = db;
  }

  async get<T extends StoreNames<S>>(
    table: T,
    id: StoreKey<S, T>
  ): Promise<StoreValue<S, T>> {
    let transaction = this.db.transaction(table, "readonly");
    let result = await transaction.store.get(id);
    if (!result) {
      throw new Error(`Key "${id}" not found in table "${table}`);
    }
```

```
198       await transaction.done;
199       return result;
200     }
201
202     async getAll<T extends StoreNames<S>>(table: T): Promise<StoreValue<S,
        ↪  T>[]> {
203       let result: StoreValue<S, T>[] = [];
204       let transaction = this.db.transaction(table, "readonly");
205       let cursor = await transaction.store.openCursor();
206       while (cursor) {
207         result.push(cursor.value);
208         cursor = await cursor.continue();
209       }
210       await transaction.done;
211       return result;
212     }
213
214     async delete<T extends StoreNames<S>>(
215       table: T,
216       id: StoreKey<S, T>
217     ): Promise<void> {
218       let transaction = this.db.transaction(table, "readwrite");
219       await transaction.store.delete(id);
220       await transaction.done;
221     }
222
223     async put<T extends StoreNames<S>>(
224       table: T,
225       id: StoreKey<S, T>,
226       value: StoreValue<S, T>
227     ): Promise<void> {
228       let transaction = this.db.transaction(table, "readwrite");
229       await transaction.store.put(value, id);
230       await transaction.done;
231     }
232
233     async close() {
234       return this.db.close();
235     }
```

```
}

export async function openDatabase(): Promise<Database> {
  let db = await openDB<DatabaseSchema>("gdb-db", 1, {
    upgrade(db, oldVersion, newVersion, transaction) {
      if (oldVersion < 1) {
        db.createObjectStore("option", { keyPath: "id" });
        db.createObjectStore("discipline", { keyPath: "id" });
        db.createObjectStore("discipline_offer", { keyPath: "id" });
        db.createObjectStore("team", { keyPath: "id" });
        db.createObjectStore("plan", { keyPath: "id" });
        db.createObjectStore("plan_version", { keyPath: "id" });
        db.createObjectStore("university", { keyPath: "id" });
        db.createObjectStore("period", { keyPath: "id" });
        db.createObjectStore("campus", { keyPath: "id" });
        db.createObjectStore("habilitation", { keyPath: "id" });
        db.createObjectStore("course", { keyPath: "id" });
        db.createObjectStore("index", { keyPath: "id" });
      }
    }
  });
  return new Database(db);
}
```

Arquivo importer.ts

```
import { Database } from "./database";
import { RemoteOfflineResponse } from "../graphql";

export default async function importData(
  database: Database,
  response: RemoteOfflineResponse
) {
  for (let disciplineOfferId of response.disciplineOffersToDelete) {
    await database.delete("discipline_offer", disciplineOfferId);
  }
  for (let disciplineId of response.disciplinesToDelete) {
    await database.delete("discipline", disciplineId);
  }
  for (let habilitationId of response.habilitationsToDelete) {
```

```
15      let habilitation = await database.get("habilitation",
        ↪  habilitationId);
16      for (let stepId of habilitation.steps) {
17        let step = await database.get("step", stepId);
18        for (let disciplineId of step.disciplines) {
19          await database.delete("discipline", disciplineId);
20        }
21        await database.delete("step", stepId);
22      }
23      await database.delete("habilitation", habilitationId);
24    }
25    for (let habilitation of response.habilitations) {
26      let steps: string[] = [];
27      for (let step of habilitation.steps) {
28        steps.push(step.id);
29        let disciplines: string[] = [];
30        for (let discipline of step.disciplines) {
31          disciplines.push(discipline.id);
32          await database.put("discipline", discipline.id, {
33            id: discipline.id,
34            code: discipline.code,
35            name: discipline.name,
36            description: discipline.description,
37            genericID: discipline.genericID,
38            related: discipline.related,
39            step: step.id,
40            course: habilitation.course.id,
41            habilitation: habilitation.id,
42            tables: discipline.tables,
43            universityID: habilitation.university.id,
44            type: discipline.type,
45            version: discipline.version
46          });
47        }
48        await database.put("step", step.id, {
49          id: step.id,
50          name: step.name,
51          disciplines
52        });
```

```
    }
    await database.put("habilitation", habilitation.id, {
      universityID: habilitation.university.id,
      id: habilitation.id,
      limits: habilitation.limits,
      name: habilitation.name,
      tables: habilitation.tables,
      version: habilitation.version,
      course: habilitation.course.id,
      steps: habilitation.steps.map(step => step.id)
    });
  }
}
```

Arquivo index.ts

```
/// <reference lib="webworker" />

import { graphql, buildSchema } from "graphql";
import schemaSource from "../schema.graphql";
import RootReducer from "./resolvers";
import { openDatabase } from "./database";
import { precacheAndRoute } from "workbox-precaching";
import { registerRoute } from "workbox-routing";

const schema = buildSchema(schemaSource);

async function graphqlProcess(request: Request) {
  let { query, variables } = await request.json();
  let db = await openDatabase();
  let result = await graphql(schema, query, new RootReducer(db),
  ↪ variables);
  await db.close();
  return new Response(JSON.stringify(result), {
    headers: {
      "Content-Type": "application/json"
    }
  });
}

```

```
function graphqlCallback({ event }: { event: FetchEvent }) {
  let copy = event.request.clone();
  return fetch(event.request).catch(() => {
    return graphqlProcess(copy);
  });
}

function main() {
  // registerRoute("/api/graphql", graphqlCallback);
  let teste = (self as any).__WB_MANIFEST;
  // precacheAndRoute((self as any).__WB_MANIFEST || []);
}

main();
```

B.2.13 Pasta src/js/pages/account

Arquivo index.tsx

```
import React from "react";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";
import View from "./components";

export default function Account() {
  const title = "Editar Conta";
  return (
    <Title title={title}>
      <AppDrawer selected="account" />
      <TopBar title={title} />
      <TopBarContent>
        <View />
      </TopBarContent>
    </Title>
  );
}
```

B.2.14 Pasta src/js/pages/create-plan

Arquivo index.tsx

```
1  import React from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6
7  export default function EditUniversity() {
8    const title = "Criar Plano";
9    return (
10     <Title title={title}>
11       <AppDrawer selected="plans" />
12       <TopBar title={title} />
13       <TopBarContent>
14         <View />
15       </TopBarContent>
16     </Title>
17   );
18 }
```

B.2.15 Pasta src/js/pages/create-university

Arquivo index.tsx

```
1  import React from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6
7  export default function CreateUniversity() {
8    const title = "Cadastrar Universidade";
9    return (
10     <Title title={title}>
11       <AppDrawer selected="universities" />
```

```
12        <TopBar title={title} />
13        <TopBarContent>
14          <View />
15        </TopBarContent>
16      </Title>
17    );
18  }
```

B.2.16 Pasta src/js/pages/discipline-offer

Arquivo index.tsx

```
1  import React from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6  import getReduxModule from "../base/store/querystring";
7  import { DynamicModuleLoader } from "redux-dynamic-modules";
8
9  export default function DisciplineOffer() {
10   const title = "Detalhes da Oferta de Disciplina";
11   return (
12     <DynamicModuleLoader modules={[getReduxModule()]}>
13       <Title title={title}>
14         <AppDrawer selected="discipline-offer" />
15         <TopBar title={title} />
16         <TopBarContent>
17           <View />
18         </TopBarContent>
19       </Title>
20     </DynamicModuleLoader>
21   );
22 }
```

B.2.17 Pasta src/js/pages/discipline

Arquivo index.tsx

```
import React from "react";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";
import View from "./components";
import getReduxModule from "../base/store/querystring";
import { DynamicModuleLoader } from "redux-dynamic-modules";

export default function Discipline() {
  const title = "Detalhes da Disciplina";
  return (
    <DynamicModuleLoader modules={[getReduxModule()]}>
      <Title title={title}>
        <AppDrawer selected="discipline" />
        <TopBar title={title} />
        <TopBarContent>
          <View />
        </TopBarContent>
      </Title>
    </DynamicModuleLoader>
  );
}
```

B.2.18 Pasta src/js/pages/edit-university

Arquivo index.tsx

```
import React from "react";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";
import View from "./components";

export default function EditUniversity() {
  const title = "Editar Universidade";
  return (
    <Title title={title}>
      <AppDrawer selected="universities" />
```

```
12        <TopBar title={title} />
13        <TopBarContent>
14          <View />
15        </TopBarContent>
16      </Title>
17    );
18  }
```

B.2.19 Pasta src/js/pages/help

Arquivo index.tsx

```
1  import React from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6  import getReduxModule from "../base/store/querystring";
7  import { DynamicModuleLoader } from "redux-dynamic-modules";
8
9  export default function Team() {
10   const title = "Ajuda";
11   return (
12     <DynamicModuleLoader modules={[getReduxModule()]}>
13       <Title title={title}>
14         <AppDrawer selected="help" />
15         <TopBar title={title} />
16         <TopBarContent>
17           <View />
18         </TopBarContent>
19       </Title>
20     </DynamicModuleLoader>
21   );
22 }
```

B.2.20 Pasta src/js/pages/home

Arquivo index.tsx

```
import React from "react";
import { useEffect } from "react";
import { useDispatch } from "react-redux";
import { redirect } from "../base/store/querystring";

export default function Home() {
  let dispatch = useDispatch();
  useEffect(
    function() {
      dispatch(
        redirect({
          url: "/planos",
          querystring: {}
        })
      );
    },
    [dispatch]
  );
  return <p>Redirecionando...</p>;
}
```

B.2.21 Pasta src/js/pages/login

Arquivo index.tsx

```
import React, { Component, ReactNode } from "react";
import View from "./components";
import { DynamicModuleLoader } from "redux-dynamic-modules";
import getReduxModule from "../base/store";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";

export default function Login() {
  const title = "Entrar";
  return (
    <DynamicModuleLoader modules={[getReduxModule()]}>
      <Title title={title}>
        <AppDrawer selected="login" />
```

```
15          <TopBar title={title} />
16          <TopBarContent>
17            <View />
18          </TopBarContent>
19        </Title>
20      </DynamicModuleLoader>
21    );
22  }
```

B.2.22 Pasta src/js/pages/offline

Arquivo index.tsx

```
1  import React, { Component, ReactNode } from "react";
2  import View from "./components";
3  import { DynamicModuleLoader } from "redux-dynamic-modules";
4  import Title from "../base/components/title";
5  import getReduxModule from "../base/store/";
6  import AppDrawer from "../base/components/partials/drawer";
7  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
8
9  export default function Offline() {
10   const title = "Acesso Offline";
11   return (
12     <DynamicModuleLoader modules={[getReduxModule()]}>
13       <Title title={title}>
14         <AppDrawer selected="offline" />
15         <TopBar title={title} />
16         <TopBarContent>
17           <View />
18         </TopBarContent>
19       </Title>
20     </DynamicModuleLoader>
21   );
22 }
```

B.2.23 Pasta src/js/pages/password-recovery

Arquivo index.tsx

```
import React, { Component, ReactNode } from "react";
import View from "./components";
import { DynamicModuleLoader } from "redux-dynamic-modules";
import getReduxModule from "../base/store/";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";

export default function PasswordRecovery() {
  const title = "Recuperar Senha";
  return (
    <DynamicModuleLoader modules={[getReduxModule()]}>
      <Title title={title}>
        <AppDrawer selected="login" />
        <TopBar title={title} />
        <TopBarContent>
          <View />
        </TopBarContent>
      </Title>
    </DynamicModuleLoader>
  );
}
```

B.2.24 Pasta src/js/pages/plan-details

Arquivo index.tsx

```
import React from "react";
import View from "./components/";
import { DynamicModuleLoader } from "redux-dynamic-modules";
import getReduxModule from "./store";

export default function PlanDetails() {
  return (
    <DynamicModuleLoader modules={[getReduxModule()]}>
      <View />
    </DynamicModuleLoader>
  );
}
```

B.2.25 Pasta src/js/pages/plan-list

Arquivo index.tsx

```
import React, { useCallback } from "react";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";
import View from "./components";
import { useDispatch } from "react-redux";
import { TopAppBarActionItem, TopAppBarSection } from
↪  "@rmwc/top-app-bar";
import { push } from "connected-react-router";

export default function PlanList() {
  const title = "Lista de Planos";
  let dispatch = useDispatch();
  let add = useCallback(() => dispatch(push("/planos/cadastrar/")),
  ↪  [dispatch]);
  return (
    <Title title={title}>
      <AppDrawer selected="plans" />

      <TopBar title={title}>
        <TopAppBarSection alignEnd>
          <TopAppBarActionItem
            icon="add"
            onClick={add}
            title="Adicionar Universidade"
          />
        </TopAppBarSection>
      </TopBar>
      <TopBarContent>
        <View />
      </TopBarContent>
    </Title>
  );
}
```

B.2.26 Pasta src/js/pages/signup

Arquivo index.tsx

```
import React from "react";
import View from "./components";
import { DynamicModuleLoader } from "redux-dynamic-modules";
import getReduxModule from "../base/store";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";

export default function Signup() {
  const title = "Criar Conta";
  return (
    <DynamicModuleLoader modules={[getReduxModule()]}>
      <Title title={title}>
        <AppDrawer selected="signup" />
        <TopBar title={title} />
        <TopBarContent>
          <View />
        </TopBarContent>
      </Title>
    </DynamicModuleLoader>
  );
}
```

B.2.27 Pasta src/js/pages/teacher

Arquivo index.tsx

```
import React from "react";
import Title from "../base/components/title";
import AppDrawer from "../base/components/partials/drawer";
import TopBar, { TopBarContent } from
↪  "../base/components/partials/top-bar";
import View from "./components";
import getReduxModule from "../base/store/querystring";
import { DynamicModuleLoader } from "redux-dynamic-modules";
```

```
8
9  export default function Teacher() {
10   const title = "Detalhes do Professor";
11   return (
12     <DynamicModuleLoader modules={[getReduxModule()]}>
13       <Title title={title}>
14         <AppDrawer selected="teacher" />
15         <TopBar title={title} />
16         <TopBarContent>
17           <View />
18         </TopBarContent>
19       </Title>
20     </DynamicModuleLoader>
21   );
22 }
```

B.2.28 Pasta src/js/pages/team

Arquivo index.tsx

```
1  import React from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6  import getReduxModule from "../base/store/querystring";
7  import { DynamicModuleLoader } from "redux-dynamic-modules";
8
9  export default function Team() {
10   const title = "Detalhes da Turma";
11   return (
12     <DynamicModuleLoader modules={[getReduxModule()]}>
13       <Title title={title}>
14         <AppDrawer selected="team" />
15         <TopBar title={title} />
16         <TopBarContent>
17           <View />
18         </TopBarContent>
19       </Title>
```

```
20      </DynamicModuleLoader>
21    );
22  }
```

B.2.29 Pasta src/js/pages/universities

Arquivo index.tsx

```
1  import React, { useCallback } from "react";
2  import Title from "../base/components/title";
3  import AppDrawer from "../base/components/partials/drawer";
4  import TopBar, { TopBarContent } from
   ↪  "../base/components/partials/top-bar";
5  import View from "./components";
6  import { useDispatch } from "react-redux";
7  import { TopAppBarActionItem, TopAppBarSection } from
   ↪  "@rmwc/top-app-bar";
8  import { push } from "connected-react-router";
9
10 export default function Universities() {
11   const title = "Universidades";
12   let dispatch = useDispatch();
13   let add = useCallback(() => dispatch(push("/universidades/cadastrar/")),
     ↪  [
14     dispatch
15   ]);
16   return (
17     <Title title={title}>
18       <AppDrawer selected="universities" />
19       <TopBar title={title}>
20         <TopAppBarSection alignEnd>
21           <TopAppBarActionItem
22             icon="add"
23             onClick={add}
24             title="Adicionar Universidade"
25           />
26         </TopAppBarSection>
27       </TopBar>
28       <TopBarContent>
29         <View />
```

```
      </TopBarContent>
    </Title>
  );
}
```

B.2.30 Pasta src/js/service-worker/indexer

Arquivo base.ts

```ts
import Hashids from "hashids";
import { ItemSet } from "./itemset";

export const metaKey = "meta";
export type MapItem = {
  bitmap: string;
  index: number;
};

export type Bitmap = {
  // Number of entities using
  count: number;
  bitmap: ItemSet;
};

export type MapKey = {
  field: string;
  value: string;
};

// SearchOptions has options related specifically to search
export type SearchOptions = {
  version: string;
  before: number;
  after: number;
  limit: number;
};

// QueryInfo has data about the page the query is requesting
export type QueryInfo = {
  version: string;
```

```
32    last: number;
33    first: number;
34    count: number;
35  };
36
37  // PageInfo has data about the page the user is requesting
38  export type PageInfo = {
39    after: number;
40    before: number;
41  };
42
43  // IndexIteratorInfo returns additional information about the iterator
44  export type IndexIteratorInfo = {
45    page: PageInfo;
46    query: QueryInfo;
47  };
48
49  // IndexIterator allows the user to iterate for all the results of the
    ↪  index
50  export interface IndexIterator {
51    info(): IndexIteratorInfo;
52    next(): Promise<{ value: any; done: boolean }>;
53  }
54
55  // Batch allows to save values directly in the index
56  export interface Batch {
57    remove(itemKey: string): void;
58    update(values: { [key: string]: string }, itemKey: string, item: any):
      ↪  void;
59    set(value: string, itemKey: string, item: any): void;
60    setField(field: string, value: string, itemKey: string, item: any):
      ↪  void;
61    commit(): void;
62  }
63
64  // ExpressionData helps Expression to get data from an index
65  export interface ExpressionData {
66    get(value: string): Promise<ItemSet>;
67    getField(field: string, value: string): Promise<ItemSet>;
```

```
68    getAll(): Promise<ItemSet>;
69  }
70
71  // Expression is a helper that defines the results which should be
    ↪  returned
72  // for the query
73  export interface Expression {
74    evaluateBitmap(data: ExpressionData): Promise<ItemSet>;
75  }
76
77  // InputDataMeta represents the data that will be sent to a Inserter
78  // so it can create a DataMeta representation of the metadata of
79  // a register in the index
80  export type InputDataMeta = {
81    key: string;
82    counter: number;
83    deleted: boolean;
84    keys: ItemSet;
85  };
86
87  // DataMeta represents the data saved about a itemKey in the index
88  export type DataMeta = {
89    counter: number;
90    deleted: boolean;
91    keys: MapKey[];
92  };
93
94  export type TreeEdge = {
95    key: string;
96    value: number;
97  };
98
99  export type TreeNode = {
100   id: number;
101   final: boolean;
102   edges: TreeEdge[];
103 };
104
```

```
// Inserter is a inserter that allows to insert meta-data from the index
↪  in
// batches - while does not knowing anything about the underlying storage
export interface Inserter {
  addData(counter: number, data: any): Promise<void>;
  commitData(): Promise<void>;
  commitTree(name: string, nodes: TreeNode[]): Promise<void>;
  commitMap(
    bitmaps: Map<string, Bitmap>,
    maps: Map<MapKey, MapItem>
  ): Promise<void>;
  commitDataMeta(
    maps: Map<MapKey, MapItem>,
    dataMetaKeys: Map<string, InputDataMeta>
  ): Promise<void>;
}

// ContextBase allows to get version and manage temporary values on the
↪  context
export interface ContextGetter {
  getVersion(): string;
  get(key: string): string;
}

// ContextSetter allows to set persistent values on the context
export interface ContextSetter extends ContextGetter {
  set(key: string, value: string): void;
}

export interface Optimizer {
  hasMapKey(ctx: ContextGetter, key: MapKey): boolean;
  hasDataMeta(ctx: ContextGetter, key: string): boolean;
  isDeleted(counter: number): boolean;
  addDataMeta(key: string): void;
  addMapKey(key: MapKey): void;
  removeDeletedKeys(bitmap: ItemSet): ItemSet;
  commit(): void;
}

```

```typescript
// Controller allows to manage all the data that is not meta for the index
export interface Controller {
  insert(ctx: ContextSetter): Inserter;
  optimize(
    sources: ContextGetter[],
    handler: Optimizer,
    target: ContextSetter
  ): Promise<void>;
  getMap(ctx: ContextGetter, key: MapKey): Promise<ItemSet | undefined>;
  getTreeNode(
    ctx: ContextGetter,
    tree: string,
    node: number
  ): Promise<TreeNode | undefined>;
  getDataMeta(
    ctx: ContextGetter,
    itemKey: string
  ): Promise<DataMeta | undefined>;
  getData(ctx: ContextGetter, counter: number): Promise<any | undefined>;
  remove(ctx: ContextGetter, toPreserve: boolean, values: string[]): void;
}

// Index works with a kv.Store to index data so queries is much more faster
// and flexible
export interface Index {
  update(): Promise<Batch>;
  rebuild(): Promise<Batch>;
  optimize(): Promise<void>;
  remove(): Promise<void>;
  runGC(): Promise<void>;
  search(expression: Expression): Promise<IndexIterator>;
  searchWithOptions(
    expression: Expression,
    options: SearchOptions
  ): Promise<IndexIterator>;
}
```

```

export const intEncoder = new Hashids("indexer package");
```

Arquivo controller.ts

```
import {
  MapKey,
  ContextSetter,
  Inserter,
  ContextGetter,
  Optimizer,
  intEncoder,
  DataMeta,
  TreeNode,
  TreeEdge,
  Controller,
  MapItem,
  Bitmap,
  InputDataMeta
} from "./base";
import { Store } from "../kv/base";
import { ItemSet } from "./itemset";

// indexMapItem is used just to host data related to an Indexer instance
↪  in the
// Store
type indexMapItem = {
  field: string;
  value: string;
  items: number[];
};

type indexTreeEdge = {
  key: string;
  value: number;
};

type indexTreeNode = {
  node: number;
  final: boolean;
```

```
35    edges: indexTreeEdge[];
36  };
37
38  // indexData is used just to host data related to an Indexer instance in
    ↪  the
39  // Store
40  type indexData = {
41    counter: number;
42    data: any;
43  };
44
45  // indexDataMeta is used just to host data related to an Indexer instance
    ↪  in the
46  // Store
47  type indexDataMeta = {
48    counter: number;
49    deleted: boolean;
50    name: string;
51    keys: MapKey[];
52  };
53
54  function kvEncoder(key: MapKey): string {
55    return key.field + "-" + key.value;
56  }
57
58  type KVOptions = {
59    TablePrefix: string;
60  };
61
62  type kvTables = {
63    mapTable: string;
64    treeTable: string;
65    dataMetaTable: string;
66    dataTable: string;
67  };
68
69  function getDataKey(version: string, counter: number): string {
70    let hash = intEncoder.encode(counter);
71    return version + "-" + hash;
```

```
}

function getMapKey(version: string, key: MapKey): string {
  let item = kvEncoder(key);
  return version + "-" + item;
}

function getTreeKey(version: string, treeName: string, nodeID: number):
↪  string {
  let hash = intEncoder.encode(nodeID);
  return version + "-" + treeName + "-" + hash;
}

function getDataMetaKey(version: string, key: string): string {
  return version + "-" + key;
}

class kvController {
  private tables: kvTables;
  private storage: Store;

  constructor(storage: Store, tables: kvTables) {
    this.storage = storage;
    this.tables = tables;
  }

  public insert(ctx: ContextSetter): Inserter {
    return new kvInserter(ctx.getVersion(), this.storage, this.tables);
  }

  private async optimizeMap(
    sources: ContextGetter[],
    handler: Optimizer,
    target: ContextSetter
  ) {
    let known = new Set<string>();
    let results = await this.storage.getAllKeys(this.tables.mapTable);
    for (let key of results) {
      let data: indexMapItem;
```

```
110       let inSource = -1;
111       let encoded: string;
112       let sourceID = 0;
113       for (let source of sources) {
114         let version = source.getVersion();
115         if (key.indexOf(version) === 0) {
116           inSource = sourceID;
117           encoded = key.substr(version.length);
118           break;
119         }
120         sourceID++;
121       }
122       if (inSource > -1) {
123         // We need to load the key we found here because we need the
          ↪  MapKey data
124         if (known.has(encoded)) {
125           // We already migrated that MapKey!
126           // Ignore it here. :)
127           continue;
128         }
129         known.add(encoded);
130         data = await this.storage.get(this.tables.mapTable, key);
131       }
132       if (inSource > -1) {
133         let mapKey: MapKey = {
134           field: data.field,
135           value: data.value
136         };
137         let resultBitmap = new ItemSet(data.items);
138         let sourceID = 0;
139         for (let source of sources) {
140           if (sourceID != inSource && handler.hasMapKey(source, mapKey))
            ↪ {
141             let newBitmap = await this.getMap(source, mapKey);
142             if (newBitmap) {
143               // Checked! Load bitmap
144               resultBitmap = resultBitmap.or(newBitmap);
145             }
146           }
```

```
        }

        resultBitmap = handler.removeDeletedKeys(resultBitmap);
        if (resultBitmap.isEmpty()) {
          // Ignore if( empty because we don't migrate deleted dat)a
          continue;
        }

        let newData: indexMapItem = {
          field: mapKey.field,
          value: mapKey.value,
          items: resultBitmap.toArray()
        };
        let newKey = getMapKey(target.getVersion(), mapKey);
        await this.storage.set(this.tables.mapTable, newKey, newData);
        handler.addMapKey(mapKey);
      }
    }
  }

  private async optimizeDataMeta(
    sources: ContextGetter[],
    handler: Optimizer,
    target: ContextSetter
  ) {
    let keys = await this.storage.getAllKeys(this.tables.dataMetaTable);
    let known = new Set<string>();
    for (let key of keys) {
      let data: indexDataMeta;
      let inSource: boolean;
      let itemKey: string;
      for (let source of sources) {
        let version = source.getVersion();
        if (key.indexOf(version) === 0) {
          inSource = true;
          itemKey = key.substr(version.length);
          break;
        }
      }
```

```
186       if (inSource) {
187         // We need to load the key we found here because we need the
            ↪ MapKey data
188         if (known.has(itemKey)) {
189           // We already migrated that DataMeta!
190           // Ignore it here. :)
191           continue;
192         }
193         known.add(itemKey);
194         data = await this.storage.get(this.tables.dataMetaTable, key);
195       }
196       if (inSource) {
197         let newKey = getDataMetaKey(target.getVersion(), data.name);
198         for (let source of sources) {
199           if (handler.hasDataMeta(source, data.name)) {
200             let oldData: DataMeta | undefined = await this.getDataMeta(
201               source,
202               data.name
203             );
204             if (oldData) {
205               // Save same data under new Key, if it's not deleted, of
                  ↪ course)
206               if (!oldData.deleted) {
207                 await this.storage.set(this.tables.dataMetaTable, newKey,
                    ↪ {
208                   Counter: oldData.counter,
209                   Deleted: oldData.deleted,
210                   Keys: oldData.keys,
211                   Name: data.name
212                 });
213
214                 await handler.addDataMeta(data.name);
215               }
216             }
217             break;
218           }
219         }
220       }
221     }
```

```
  }

  private async optimizeData(
    sources: ContextGetter[],
    handler: Optimizer,
    target: ContextSetter
  ) {
    let keys = await this.storage.getAllKeys(this.tables.dataTable);
    for (let key of keys) {
      var inSource: boolean;
      for (let source of sources) {
        let version = source.getVersion();
        if (key.indexOf(version) === 0) {
          inSource = true;
          break;
        }
      }
      if (inSource) {
        let data: indexData = await this.storage.get(
          this.tables.dataTable,
          key
        );
        if (!handler.isDeleted(data.counter)) {
          var newKey: string;
          newKey = getDataKey(target.getVersion(), data.counter);
          // Save same data under new Key

          await this.storage.set(this.tables.dataTable, newKey, data);
        }
      }
    }
  }

  async optimize(
    sources: ContextGetter[],
    handler: Optimizer,
    target: ContextSetter
  ) {
    await this.optimizeMap(sources, handler, target);
```

```
    await this.optimizeDataMeta(sources, handler, target);
    await this.optimizeData(sources, handler, target);
    await handler.commit();
  }

  async getMap(ctx: ContextGetter, key: MapKey): Promise<ItemSet |
  ↪ undefined> {
    let dbKey = getMapKey(ctx.getVersion(), key);
    let data: indexMapItem = await this.storage.get(
      this.tables.mapTable,
      dbKey
    );
    if (data && data.field == key.field && data.value == key.value) {
      return new ItemSet(data.items);
    }
  }

  async getDataMeta(ctx: ContextGetter, itemKey: string):
  ↪ Promise<DataMeta> {
    let dbKey = getDataMetaKey(ctx.getVersion(), itemKey);
    let data: indexDataMeta = await this.storage.get(
      this.tables.dataMetaTable,
      dbKey
    );
    if (data && data.name == itemKey) {
      return {
        counter: data.counter,
        deleted: data.deleted,
        keys: data.keys
      };
    }
  }

  async getTreeNode(
    ctx: ContextGetter,
    name: string,
    nodeID: number
  ): Promise<TreeNode | undefined> {
    let key = getTreeKey(ctx.getVersion(), name, nodeID);
```

```
    var dbResult: indexTreeNode = await this.storage.get(
      this.tables.treeTable,
      key
    );
    if (dbResult && dbResult.node == nodeID) {
      let edges: TreeEdge[] = [];
      for (let edge of dbResult.edges) {
        edges.push({
          key: edge.key,
          value: edge.value
        });
      }
      return {
        edges,
        final: dbResult.final,
        id: nodeID
      };
    }
  }

  getData(ctx: ContextGetter, counter: number): Promise<any> {
    let key = getDataKey(ctx.getVersion(), counter);
    return this.storage.get(this.tables.dataTable, key);
  }

  async deleteTable(
    ctx: ContextGetter,
    table: string,
    toPreserve: boolean,
    values: string[]
  ) {
    let keys = await this.storage.getAllKeys(table);
    for (let key of keys) {
      if (key.indexOf(ctx.getVersion()) === 0) {
        let hasPrefix = false;
        for (let check of values) {
          if (key.indexOf(check)) {
            hasPrefix = true;
            break;
```

```
            }
          }
          if (!hasPrefix === toPreserve) {
            await this.storage.del(table, key);
          }
        }
      }
    }

    async remove(ctx: ContextGetter, toPreserve: boolean, values: string[])
    ↪ {
      // Used by Delete and RunGC methods in Index
      await this.deleteTable(ctx, this.tables.mapTable, toPreserve,
      ↪ values);
      await this.deleteTable(ctx, this.tables.treeTable, toPreserve,
      ↪ values);
      await this.deleteTable(ctx, this.tables.dataMetaTable, toPreserve,
      ↪ values);
      await this.deleteTable(ctx, this.tables.dataTable, toPreserve,
      ↪ values);
    }
}

function NewKVController(storage: Store, options: KVOptions): Controller
↪ {
  return new kvController(storage, {
    dataMetaTable: options.TablePrefix + "-data-meta",
    dataTable: options.TablePrefix + "-data",
    mapTable: options.TablePrefix + "-map",
    treeTable: options.TablePrefix + "-tree"
  });
}

type kvDataBatchItem = {
  counter: number;
  start: number;
  end: number;
};

```

```
type kvMapBatchItem = {
  start: number;
  end: number;
};

export class kvInserter {
  version: string;
  storage: Store;
  tables: kvTables;

  constructor(version: string, storage: Store, tables: kvTables) {
    this.version = version;
    this.storage = storage;
    this.tables = tables;
  }

  async addData(counter: number, data: any) {
    let key = getDataKey(this.version, counter);
    await this.storage.set(this.tables.dataTable, key, {
      counter,
      data
    });
  }

  commitData(): Promise<void> {
    return;
  }

  async commitMap(
    bitmaps: Map<string, Bitmap>,
    keys: Map<MapKey, MapItem>
  ): Promise<void> {
    for (let [key, value] of keys.entries()) {
      let encodedKey = getMapKey(this.version, key);
      await this.storage.set(this.tables.mapTable, encodedKey, {
        items: bitmaps.get(value.bitmap).bitmap,
        field: key.field,
        value: key.value
      });
```

```
409     }
410   }
411
412   async commitTree(name: string, tree: TreeNode[]) {
413     let nodeID = 0;
414     for (let node of tree) {
415       let encodedKey = getTreeKey(this.version, name, nodeID);
416       let edges: indexTreeEdge[] = Array(node.edges.length);
417       let edgeID = 0;
418       for (let edge of node.edges) {
419         edges[edgeID] = { key: edge.key, value: edge.value };
420         edgeID++;
421       }
422       await this.storage.set(this.tables.treeTable, encodedKey, {
423         node: nodeID,
424         edges,
425         final: node.final
426       });
427     }
428   }
429
430   async commitDataMeta(
431     keys: Map<MapKey, MapItem>,
432     dataMetaKeys: Map<string, InputDataMeta>
433   ) {
434     let keyList: MapKey[] = Array(keys.size);
435     for (let [key, value] of keys.entries()) {
436       keyList[value.index] = key;
437     }
438     for (let [itemKey, meta] of dataMetaKeys.entries()) {
439       let dbKey = getDataMetaKey(this.version, itemKey);
440       let result: indexDataMeta = {
441         name: itemKey,
442         counter: meta.counter,
443         deleted: meta.deleted,
444         keys: []
445       };
446       let metaKeys = meta.keys.toArray();
447       for (let index of metaKeys) {
```

```
448         let key = keyList[index];
449         result.keys.push(key);
450       }
451       await this.storage.set(this.tables.dataMetaTable, dbKey, result);
452     }
453   }
454 }
```

Arquivo exact.ts

```
1  import {
2    ContextSetter,
3    Index,
4    Controller,
5    Batch,
6    metaKey,
7    Optimizer,
8    ContextGetter,
9    SearchOptions,
10   IndexIteratorInfo,
11   Expression,
12   IndexIterator
13 } from "./base";
14 import { Store } from "../kv/base";
15 import { ExactOptions, getDefaultExactOptions } from "./exact_options";
16 import {
17   exactMetaVersions,
18   includesVersion,
19   getSourcesVersion
20 } from "./exact_models";
21 import { exactContext } from "./exact_context";
22 import { getItemsBitmaps } from "./utilities";
23 import { exactIterator } from "./exact_iterator";
24 import { exactBatch } from "./exact_batch";
25 import { exactResolver } from "./exact_resolver";
26 import { ItemSet } from "./itemset";
27 import exactOptimizer from "./exact_optimizer";
28
29 export function getMetaTable(options: ExactOptions): string {
30   return options.metaTable;
```

```
}

export function getMetaKey(options: ExactOptions): string {
  return options.name + "-" + metaKey;
}

export class exactIndexer implements Index {
  protected storage: Store; // t stores data for the index
  protected exactOptions: ExactOptions;
  protected controller: Controller;

  constructor(
    storage: Store,
    exactOptions: ExactOptions,
    controller: Controller
  ) {
    this.storage = storage;
    this.exactOptions = getDefaultExactOptions(exactOptions);
    this.controller = controller;
  }

  async rebuild(): Promise<Batch> {
    return this.newExactBatch();
  }

  async update(): Promise<Batch> {
    let old: exactMetaVersions | undefined = await this.storage.get(
      getMetaTable(this.exactOptions),
      getMetaKey(this.exactOptions)
    );
    if (!old) {
      return this.rebuild();
    }
    return this.newExactBatch(old);
  }

  protected async generateNewVersionID(): Promise<string> {
    let version = await this.storage.generateID(
      getMetaTable(this.exactOptions)
```

```
    );
    return this.exactOptions.name + "-" + version;
  }

  protected async newExactBatch(old?: exactMetaVersions):
  ↪  Promise<exactBatch> {
    let version = await this.generateNewVersionID();
    let allItems = new ItemSet();
    if (old && old.versions.length > 0) {
      allItems = new ItemSet(old.versions[0].allItems);
    }
    let items = new ItemSet();
    let ctx = new exactContext(version, {});
    return new exactBatch(
      ctx,
      allItems,
      items,
      old,
      this,
      this.storage,
      this.exactOptions,
      this.controller
    );
  }

  protected newExactOptimizer(
    old: exactMetaVersions,
    ctx: exactContext
  ): Optimizer {
    let allItems = new ItemSet(old.versions[0].allItems);
    return new exactOptimizer(
      old,
      ctx,
      allItems,
      this.storage,
      this.exactOptions
    );
  }

```

```
  async optimize(): Promise<void> {
    let old: exactMetaVersions = await this.storage.get(
      getMetaTable(this.exactOptions),
      getMetaKey(this.exactOptions)
    );
    if (!old) {
      return;
    }
    if (old.versions.length >= this.exactOptions.mergeMinVersions) {
      let version: string = await this.generateNewVersionID();
      let target = new exactContext(version, {});
      let optimizer = this.newExactOptimizer(old, target);
      let sources = getSourcesVersion(old);
      await this.controller.optimize(sources, optimizer, target);
      await this.runGC();
    }
  }
  async remove(): Promise<void> {
    let metaTable = getMetaTable(this.exactOptions);
    let keys = await this.storage.getAllKeys(metaTable);
    let name = this.exactOptions.name + "-";
    for (let key of keys) {
      if (key.indexOf(name) === 0) {
        await this.storage.del(metaTable, key);
      }
    }
    this.controller.remove(new exactContext(name + "-", {}), false, [
      name + "-"
    ]);
  }

  protected async getResolverData(
    version: string
  ): Promise<[ItemSet, exactMetaVersions]> {
    if (version == "" || version == "latest") {
      version = this.exactOptions.name + "-" + metaKey;
    }
    let meta: exactMetaVersions = await this.storage.get(
      getMetaTable(this.exactOptions),
```

```
      version
    );
    let all = new ItemSet();
    if (meta && meta.versions.length > 0) {
      all = new ItemSet(meta.versions[0].allItems);
    }
    return [all, meta];
  }

  getIterator(
    meta: exactMetaVersions,
    result: ItemSet | null,
    options: SearchOptions
  ): [number[], IndexIteratorInfo] {
    let info: IndexIteratorInfo = {
      query: {
        count: result.cardinality(),
        version: "latest",
        first: result.isEmpty() ? 0 : result.lsb(),
        last: result.isEmpty() ? 0 : result.msb()
      },
      page: {
        after: 0,
        before: 0
      }
    };
    if (options.before > 0) {
      // Remove all the values AFTER the before value
      result.setRange(options.before, 1 << 32, 0);
      if (options.limit > 0 && result.cardinality() > options.limit) {
        let min = result.cardinality() - options.limit;
        let first: number = result.select(min);
        // And then remove the values BEFORE the window we want to see..
        result.setRange(0, first, 0);
      }
    } else {
      result.setRange(0, options.after, 1); // Remove from start up to a
      ↪ value
      if (options.limit > 0 && result.cardinality() > options.limit) {
```

```
        let last: number = result.select(options.limit);
        // Remove the records AFTER the window we want to see
        result.setRange(last, 1 << 32, 0);
      }
    }
    if (!result.isEmpty()) {
      info.page.before = result.lsb();
      info.page.after = result.msb() + 1;
    }
    return [result.toArray(), info];
  }

  search(expression: Expression): Promise<IndexIterator> {
    return this.searchWithOptions(expression);
  }

  async searchWithOptions(
    expression: Expression,
    options?: SearchOptions
  ): Promise<IndexIterator> {
    let [all, meta] = await this.getResolverData(options?.version ?? "");
    let data = new exactResolver(all, meta, this.controller,
    ↪  this.exactOptions);
    let result = await expression.evaluateBitmap(data);
    let [results, info] = this.getIterator(meta, result, options);
    return new exactIterator(results, this.controller, meta, info);
  }

  async runGC(): Promise<void> {
    let metaTable = getMetaTable(this.exactOptions);
    let indexMetaKey = getMetaKey(this.exactOptions);
    let name = this.exactOptions.name + "-";
    let meta: exactMetaVersions = await this.storage.get(
      metaTable,
      indexMetaKey
    );
    let it = await this.storage.getAllKeys(metaTable);
    var keys: string[] = [];
    for (let key of it) {
```

```
      if (
        key.indexOf(name) !== 0 ||
        includesVersion(meta, key) ||
        key === indexMetaKey
      ) {
        continue;
      }
      keys.push(key);
    }
    for (let key of keys) {
      await this.storage.del(metaTable, key);
    }
    // We only invoke the controller's delete method IF we have deleted some version key
    // on the index. If, for some reason, the controller's delete method cannot be completed,
    // as the index grows and new versions appear the index will be cleaned again completely,
    // without problems.
    if (keys.length > 0) {
      let versions: string[] = [];
      for (let { version } of meta.versions) {
        versions.push(version);
      }
      this.controller.remove(new exactContext(name, {}), true, versions);
    }
  }
}
```

Arquivo exact_batch.ts

```
import {
  Inserter,
  ContextSetter,
  ContextGetter,
  DataMeta,
  Controller,
  InputDataMeta,
  MapKey
} from "./base";
```

```
import {
  exactMetaVersions,
  exactMetaVersion,
  getSourcesVersion
} from "./exact_models";
import { exactIndexer, getMetaTable, getMetaKey } from "./exact";
import { LimitedStore, Store } from "../kv/base";
import { ExactOptions } from "./exact_options";
import { exactContext } from "./exact_context";
import { ItemSet } from "./itemset";

type MapPosition = {
  bitmap: string;
  index: number;
};

type Bitmap = {
  // Number of entities using
  count: number;
  bitmap: ItemSet;
};

type exactFieldMap = {
  bitmap: string;
  index: number;
  fieldLength: number;
};

export class exactBatch {
  protected ctx: exactContext;
  protected inserter: Inserter;
  protected allItems: ItemSet;
  protected items: ItemSet;
  protected sources: ContextGetter[];
  protected old: exactMetaVersions;
  protected newItemsCount: number;
  protected committed: boolean;
  protected exact: exactIndexer;
  protected storage: Store;
```

```
  protected exactOptions: ExactOptions;
  protected bitmaps: Map<string, Bitmap>;
  protected keys: Map<MapKey, MapPosition>;
  protected oldItems: Set<string>;
  protected datametas: Map<string, InputDataMeta>;
  protected controller: Controller;

  constructor(
    ctx: exactContext,
    allItems: ItemSet,
    items: ItemSet,
    old: exactMetaVersions,
    exact: exactIndexer,
    storage: Store,
    exactOptions: ExactOptions,
    controller: Controller
  ) {
    this.ctx = ctx;
    this.inserter = controller.insert(ctx);
    this.allItems = allItems;
    this.items = items;
    this.exact = exact;
    this.storage = storage;
    this.exactOptions = exactOptions;
    this.sources = getSourcesVersion(old);
    this.newItemsCount = 0;
    this.committed = false;
    this.bitmaps = new Map<string, Bitmap>();
    this.keys = new Map<MapKey, MapPosition>();
    this.oldItems = new Set<string>();
    this.datametas = new Map<string, InputDataMeta>();
    this.controller = controller;
  }

  protected bitmapEncode(bitmap: ItemSet): string {
    return bitmap.toString();
  }

  private bitmapAdd(key: string, x: number): string {
```

```
    let entry = this.bitmaps.get(key);
    if (!!entry) {
      if (!entry.bitmap.get(x) && entry.bitmap.set(x, 1)) {
        let newKey = this.bitmapEncode(entry.bitmap);
        entry.count--;
        if (entry.count > 0) {
          this.bitmaps[key] = entry;
        } else {
          this.bitmaps.delete(key);
        }
        key = newKey;
        let newEntry = this.bitmaps.get(key);
        if (!!newEntry) {
          if (newEntry.bitmap.equals(entry.bitmap)) {
            newEntry.count++;
            this.bitmaps.set(key, newEntry);
            if (entry.count == 0) {
              entry.bitmap.clear();
            } else {
              entry.bitmap.set(x, 0);
            }
          }
        } else if (entry.count > 0) {
          let newBitmap = entry.bitmap.clone();
          entry.bitmap.set(x, 0);
          this.bitmaps.set(key, {
            count: 1,
            bitmap: newBitmap
          });
        } else {
          entry.count++;
          this.bitmaps.set(key, entry);
        }
      }
    }
    return key;
  }

  private bitmapInitialize(x: number): string {
```

```
    let bitmap = new ItemSet([x]);
    let key = this.bitmapEncode(bitmap);
    let entry = this.bitmaps.get(key);
    if (!!entry) {
      entry.count++;
      this.bitmaps.set(key, entry);
    } else {
      this.bitmaps.set(key, {
        bitmap: bitmap,
        count: 1
      });
    }
    return key;
  }

  private async loadOldItem(itemKey: string) {
    if (this.old.versions.length === 0 || this.oldItems.has(itemKey)) {
      return;
    }
    this.oldItems.add(itemKey);
    for (let source of this.sources) {
      let meta: DataMeta = await this.controller.getDataMeta(source,
      ↪  itemKey);
      if (!!meta) {
        let keys = new ItemSet();
        for (let key of meta.keys) {
          var newBitmap: string;
          let entry = this.keys.get(key);
          if (!entry) {
            newBitmap = this.bitmapInitialize(meta.counter);
            entry = { index: this.keys.size, bitmap: "" };
          } else {
            newBitmap = this.bitmapAdd(entry.bitmap, meta.counter);
          }
          if (newBitmap != "" && newBitmap != entry.bitmap) {
            entry.bitmap = newBitmap;
            this.keys.set(key, entry);
          }
          keys.set(entry.index, 1);
```

```
        }
        this.datametas.set(itemKey, {
          key: itemKey,
          counter: meta.counter,
          deleted: meta.deleted,
          keys
        });
        break;
      }
    }
  }

  async remove(itemKey: string) {
    if (!this.committed && this.old.versions.length > 0) {
      await this.loadOldItem(itemKey);
      var exists: boolean;
      var meta: InputDataMeta | undefined = this.datametas.get(itemKey);
      if (!!meta && exists && this.items.get(meta.counter)) {
        throw new Error("Cannot delete");
      } else if (!exists || meta.deleted) {
        return;
      }
      meta.deleted = true;
      meta.keys.clear();
      this.allItems.set(meta.counter, 0);
      this.datametas.set(itemKey, meta);
    }
  }

  public async update(
    values: { [key: string]: string },
    itemKey: string,
    item: any
  ) {
    await this.remove(itemKey);
    for (let [field, value] of Object.entries(values)) {
      await this.setField(field, value, itemKey, item);
    }
  }
```

```
204
205   public async set(value: string, itemKey: string, item: any) {
206     return this.setField(this.exactOptions.defaultField, value, itemKey,
      ↪  item);
207   }
208
209   private getNextID(): number {
210     var min: number;
211     if (this.old.versions.length > 0) {
212       // The old max is now the new minimum
213       min = this.old.versions[0].max;
214     }
215     return this.newItemsCount + min;
216   }
217
218   public async setField(
219     field: string,
220     value: string,
221     itemKey: string,
222     item: any
223   ) {
224     if (!this.committed) {
225       // Load any old content on the bitmap so we do not overwrite
        ↪  anything
226       if (this.old.versions.length > 0) {
227         await this.loadOldItem(itemKey);
228       }
229       let key: MapKey = { field, value };
230       let dataMetaEntry = this.datametas.get(itemKey);
231       if (!dataMetaEntry || dataMetaEntry.deleted) {
232         this.newItemsCount++;
233         this.datametas.set(itemKey, {
234           deleted: false,
235           key: itemKey,
236           counter: this.getNextID(),
237           keys: new ItemSet()
238         });
239       }
240       // Key was set! So just mark it as NOT deleted
```

```
      let { counter } = dataMetaEntry;
      var keyEntry: MapPosition | undefined = this.keys.get(key);
      var newBitmap: string;
      if (!keyEntry) {
        newBitmap = this.bitmapInitialize(counter);
        keyEntry = { index: this.keys.size, bitmap: "" };
      } else if (
        !dataMetaEntry.keys.get(keyEntry.index) &&
        dataMetaEntry.keys.set(keyEntry.index, 1)
      ) {
        newBitmap = this.bitmapAdd(keyEntry.bitmap, counter);
      }
      if (newBitmap != "" && newBitmap != keyEntry.bitmap) {
        dataMetaEntry.keys.set(keyEntry.index, 1);
        keyEntry.bitmap = newBitmap;
        this.keys.set(key, keyEntry);
      }
      this.datametas.set(itemKey, dataMetaEntry);
      let isNew = !this.allItems.get(counter) &&
      ↪  this.allItems.set(counter, 1);
      this.items.set(counter, 1);
      if (isNew) {
        await this.inserter.addData(counter, item);
      }
    }
  }

  public async commit() {
    await this._commit();
    let autoMerge = this.exactOptions.autoMergeMinVersions;
    let oldVersionsLen = this.old.versions.length + 1;
    if (autoMerge > 0 && autoMerge > oldVersionsLen) {
      await this.exact.optimize();
    }
  }

  protected async _commit(callback?: () => void) {
    this.committed = true;
    await this.inserter.commitData();
```

```
    if (!!callback) {
      callback();
    }
    for (let [key, entry] of this.bitmaps.entries()) {
      entry.bitmap = entry.bitmap.and(this.allItems);
      this.bitmaps.set(key, entry);
    }
    await this.inserter.commitMap(this.bitmaps, this.keys);
    await this.inserter.commitDataMeta(this.keys, this.datametas);
    // If all went OK, we can update the meta table to have the right
    ↪  version
    let metaTable = getMetaTable(this.exactOptions);
    let indexMetaKey = getMetaKey(this.exactOptions);
    await this.storage.transaction([metaTable], async (db: LimitedStore)
    ↪  => {
      let oldMeta: exactMetaVersions = await db.get(metaTable,
      ↪  indexMetaKey);
      if (this.old.versions.length > 0) {
        let versions1 = oldMeta.versions;
        let versions2 = this.old.versions;
        let conflict = versions1.length !== versions2.length;
        if (!conflict) {
          let count = 0;
          for (let i = 0; i < versions1.length; i++) {
            if (versions1[i].version == versions2[i].version) {
              count++;
            }
          }
          conflict = count !== versions2.length;
        }
        if (conflict) {
          throw new Error("Conflict detected");
        }
      }
      let metaVersion: exactMetaVersion = {
        version: this.ctx.getVersion(),
        max: this.getNextID(),
        allItems: this.allItems.toArray(),
        items: this.items.toArray()
```

```
315       };
316       let newMeta: exactMetaVersions = {
317         config: [],
318         versions: []
319       };
320       newMeta.versions.push(metaVersion);
321       newMeta.versions.push(...this.old.versions);
322       this.ctx.onEach(function(key, value) {
323         newMeta.config.push({
324           version: this.ctx.GetVersion(),
325           key: key,
326           value: value
327         });
328       });
329       newMeta.config.push(...this.old.config);
330       await db.set(metaTable, this.ctx.getVersion(), newMeta);
331       await db.set(metaTable, indexMetaKey, newMeta);
332     });
333   }
334 }
```

Arquivo exact_context.ts

```
1  import { ContextSetter } from "./base";
2
3  export class exactContext implements ContextSetter {
4    private config: { [key: string]: string };
5    private version: string;
6
7    constructor(version: string, config: { [key: string]: string }) {
8      this.version = version;
9      this.config = config;
10   }
11
12   getVersion(): string {
13     return this.version;
14   }
15
16   get(key: string): string {
17     return this.config[key];
```

```
  }

  set(key: string, value: string) {
    this.config[key] = value;
  }

  onEach(callback: (key: string, value: string) => void) {
    for (let [key, value] of Object.entries(this.config)) {
      callback(key, value);
    }
  }
}
```

Arquivo exact_iterator.ts

```
import { Controller, IndexIteratorInfo } from "./base";
import { exactMetaVersions, getContextVersion } from "./exact_models";
import { getItemsBitmaps } from "./utilities";
import { ItemSet } from "./itemset";

export class exactIterator {
  private results: number[];
  private controller: Controller;
  private cursor: number;
  meta: exactMetaVersions;
  items: ItemSet[];
  infoData: IndexIteratorInfo;

  constructor(
    results: number[],
    controller: Controller,
    meta: exactMetaVersions,
    info: IndexIteratorInfo
  ) {
    this.results = results;
    this.controller = controller;
    this.cursor = 0;
    this.meta = meta;
    this.items = getItemsBitmaps(meta);
    this.infoData = info;
```

```
  }

  public info(): IndexIteratorInfo {
    return this.infoData;
  }

  public async next(): Promise<{ value: any; done: boolean }> {
    if (this.results.length <= this.cursor) {
      return { value: null, done: true };
    }
    var version: string;
    let index = this.results[this.cursor];
    let i = 0;
    for (let data of this.items) {
      if (data.get(index)) {
        version = this.meta.versions[i].version;
        break;
      }
      i++;
    }
    let ctx = getContextVersion(this.meta, version);
    let value = await this.controller.getData(ctx, index);

    this.cursor++;
    return { value, done: false };
  }
}
```

Arquivo exact_models.ts

```
import { ContextGetter } from "./base";
import { exactContext } from "./exact_context";

// exactMetaVersion should be ideally considered private: It is used just
↪  to
// host data related to an exactIndexer instance in the Store
export type exactMetaVersion = {
  version: string;
  allItems: number[]; // Bitmap
  items: number[]; // Bitmap
```

```
  max: number;
};

export type exactMetaConfig = {
  version: string;
  key: string;
  value: string;
};

// exactMetaVersions should be ideally considered private: It is used
↪  just to
// host data related to an exactIndexer instance in the Store
export type exactMetaVersions = {
  versions: exactMetaVersion[];
  config: exactMetaConfig[];
};

export function includesVersion(
  versions: exactMetaVersions,
  key: string
): boolean {
  return findVersion(versions, key) > -1;
}
export function findVersion(
  { versions }: exactMetaVersions,
  key: string
): number {
  let result = -1;
  let i = 0;
  for (let { version } of versions) {
    if (key.indexOf(version) === 0) {
      result = i;
      break;
    }
  }
  return;
}

export function getContextVersion(
```

```
  { config }: exactMetaVersions,
  version: string
): ContextGetter {
  let newConfig: { [key: string]: string } = {};
  for (let item of config) {
    if (item.version == version) {
      newConfig[item.key] = item.value;
    }
  }
  return new exactContext(version, newConfig);
}

export function getSourcesVersion(old: exactMetaVersions):
↪  ContextGetter[] {
  let sources: ContextGetter[] = [];
  for (let { version } of old.versions) {
    sources.push(getContextVersion(old, version));
  }
  return sources;
}
```

Arquivo exact_optimizer.ts

```
import { exactMetaVersions, exactMetaVersion } from "./exact_models";
import { ContextSetter, ContextGetter, MapKey } from "./base";
import { exactIndexer, getMetaKey, getMetaTable } from "./exact";
import { LimitedStore, Store } from "../kv/base";
import { ExactOptions } from "./exact_options";
import { exactContext } from "./exact_context";
import { ItemSet } from "./itemset";

export default class exactOptimizer {
  private allItems: ItemSet;
  private old: exactMetaVersions;
  private ctx: exactContext;
  private storage: Store;
  private options: ExactOptions;

  constructor(
    old: exactMetaVersions,
```

```
    ctx: exactContext,
    allItems: ItemSet,
    storage: Store,
    options: ExactOptions
  ) {
    this.allItems = allItems;
    this.old = old;
    this.ctx = ctx;
    this.storage = storage;
    this.options = options;
  }

  hasMapKey(ctx: ContextGetter, key: MapKey): boolean {
    return true;
  }

  hasDataMeta(ctx: ContextGetter, key: string): boolean {
    return true;
  }

  isDeleted(counter: number): boolean {
    return !this.allItems.get(counter);
  }

  addDataMeta(key: string): void {}

  addMapKey(key: MapKey): void {}

  removeDeletedKeys(bitmap: ItemSet): ItemSet {
    return bitmap.and(this.allItems);
  }

  async commit(): Promise<void> {
    let metaTable = getMetaTable(this.options);
    let indexMetaKey = getMetaKey(this.options);
    await this.storage.transaction([metaTable], async (db: LimitedStore)
    ↪  => {
      let oldMeta: exactMetaVersions = await db.get(metaTable,
      ↪  indexMetaKey);
```

```
55        let versions1 = oldMeta.versions;
56        let versions2 = this.old.versions;
57        let conflict = versions1.length !== versions2.length;
58        if (!conflict) {
59          let count = 0;
60          for (let i = 0; i < versions1.length; i++) {
61            if (versions1[i].version == versions2[i].version) {
62              count++;
63            }
64          }
65          conflict = count !== versions2.length;
66        }
67        if (conflict) {
68          throw new Error("Conflict detected");
69        }
70        let newMeta: exactMetaVersions = {
71          config: [],
72          versions: []
73        };
74
75        let metaVersion: exactMetaVersion = {
76          version: this.ctx.getVersion(),
77          allItems: this.allItems.toArray(),
78          items: this.allItems.toArray(),
79          max: this.old.versions[0].max
80        };
81        newMeta.versions.push(metaVersion);
82        this.ctx.onEach(function(key, value) {
83          newMeta.config.push({
84            version: this.ctx.GetVersion(),
85            key,
86            value
87          });
88        });
89        await db.set(metaTable, this.ctx.getVersion(), newMeta);
90        await db.set(metaTable, indexMetaKey, newMeta);
91      });
92    }
93  }
```

Arquivo exact_options.ts

```
// ExactOptions define a few options that are customizable on
↪  ExactIndexer
export type ExactOptions = {
  name: string;
  defaultField: string;
  metaTable: string;
  falsePositiveKeys: number;
  mergeMinVersions: number;
  autoMergeMinVersions: number;
};

export function getDefaultExactOptions(
  options: Partial<ExactOptions>
): ExactOptions {
  let name = options.name ?? "";
  return {
    name,
    defaultField: options.defaultField ?? "default",
    mergeMinVersions:
      !options.mergeMinVersions || options.mergeMinVersions < 2
        ? 2
        : options.mergeMinVersions,
    autoMergeMinVersions: options.autoMergeMinVersions ?? 0,
    falsePositiveKeys:
      !options.falsePositiveKeys || options.falsePositiveKeys < 0
        ? 0.01
        : options.falsePositiveKeys,
    metaTable: name + "-meta"
  };
}
```

Arquivo exact_resolver.ts

```
import { ExactOptions } from "./exact_options";
import { exactMetaVersions, getContextVersion } from "./exact_models";
import { MapKey, Controller, ExpressionData } from "./base";
import { ItemSet } from "./itemset";

```

```
export class exactResolver implements ExpressionData {
  protected all: ItemSet;
  protected meta: exactMetaVersions;
  protected controller: Controller;
  protected exactOptions: ExactOptions;

  constructor(
    all: ItemSet,
    meta: exactMetaVersions,
    controller: Controller,
    exactOptions: ExactOptions
  ) {
    this.all = all;
    this.meta = meta;
    this.controller = controller;
    this.exactOptions = exactOptions;
  }

  get(value: string): Promise<ItemSet> {
    return this.getField(this.exactOptions.defaultField, value);
  }

  async getField(field: string, value: string): Promise<ItemSet> {
    let mapKey: MapKey = { field, value };
    let bitmap = new ItemSet();
    for (let { version } of this.meta.versions) {
      let ctx = getContextVersion(this.meta, version);
      let map = await this.controller.getMap(ctx, mapKey);
      if (!!map) {
        bitmap = bitmap.or(map);
      }
    }
    if (!bitmap.isEmpty()) {
      bitmap = bitmap.and(this.all);
    }
    return bitmap;
  }

  async getAll(): Promise<ItemSet> {
```

```
45      return this.all.clone();
46    }
47  }
```

Arquivo expression.ts

```
1   import { Expression, ExpressionData } from "./base";
2   import { ItemSet } from "./itemset";
3
4   // All returns all results from the index
5   class All implements Expression {
6     // evaluateBitmap returns the bitmap with the result IDs
7     evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
8       return data.getAll();
9     }
10  }
11
12  // And returns the intersection between the results of the expressions
13  class And implements Expression {
14    private operations: Expression[];
15    // And returns a new AND expression
16    constructor(...operations: Expression[]) {
17      this.operations = operations;
18    }
19
20    // evaluateBitmap returns the bitmap with the result IDs
21    async evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
22      var result: ItemSet;
23      for (let operation of this.operations) {
24        let bitmap = await operation.evaluateBitmap(data);
25        if (bitmap === null) {
26          bitmap = result;
27        }
28        if (result === null) {
29          result = bitmap;
30        } else {
31          result = result.and(bitmap);
32        }
33        if (result === null || result.isEmpty()) {
34          // Short-Circuit to stop loop when there were no results already
```

```
35          break;
36        }
37      }
38      return result;
39    }
40  }
41
42  // Or returns the union between the results of the expressions
43  class Or implements Expression {
44    private operations: Expression[];
45
46    // Returns a new OR expression
47    constructor(...operations: Expression[]) {
48      this.operations = operations;
49    }
50
51    // evaluateBitmap returns the bitmap with the result IDs
52    async evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
53      var result: ItemSet;
54      for (let operation of this.operations) {
55        var bitmap: ItemSet = await operation.evaluateBitmap(data);
56        if (bitmap === null) {
57          bitmap = result;
58        }
59        if (result === null) {
60          result = bitmap;
61        } else {
62          result = result.or(bitmap);
63        }
64      }
65      return result;
66    }
67  }
68
69  // Not returns all results that are not accepted by the Expression
70  class Not implements Expression {
71    private operation: Expression;
72
73    constructor(operation: Expression) {
```

```
    this.operation = operation;
  }

  // evaluateBitmap returns the bitmap with the result IDs
  async evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
    let result: ItemSet = await data.getAll();
    let bitmap: ItemSet = await this.operation.evaluateBitmap(data);
    if (bitmap != null) {
      result = result.andNot(bitmap);
    }
    return result;
  }
}

// Terminal returns all results that match the specified terminal with
↪ default
// field
class Terminal implements Expression {
  private value: string;
  constructor(value: string) {
    this.value = value;
  }

  // evaluateBitmap returns the bitmap with the result IDs
  evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
    return data.get(this.value);
  }
}

// FieldTerminal returns all results that match the specified terminal
↪ with
// specified field
class FieldTerminal implements Expression {
  attribute: string;
  value: string;
  constructor(attribute: string, value: string) {
    this.attribute = attribute;
    this.value = value;
  }
```

```
111   // evaluateBitmap returns the bitmap with the result IDs
112   evaluateBitmap(data: ExpressionData): Promise<ItemSet> {
113     return data.getField(this.attribute, this.value);
114   }
115 }
```

Arquivo fulltext.ts

```
1  import { exactIndexer, getMetaTable, getMetaKey } from "./exact";
2  import { exactMetaVersions, getSourcesVersion } from "./exact_models";
3  import { ExactOptions } from "./exact_options";
4  import {
5    Controller,
6    Batch,
7    ContextGetter,
8    Optimizer,
9    Expression,
10   IndexIterator,
11   SearchOptions
12 } from "./base";
13 import { Store } from "../kv/base";
14 import { getDefaultFullTextOptions, FullTextOptions } from
   ↪  "./fulltext_options";
15 import ftBatch from "./fulltext_batch";
16 import { exactContext } from "./exact_context";
17 import { ItemSet } from "./itemset";
18 import ftOptimizer from "./fulltext_optimizer";
19 import { ftResolver } from "./fulltext_resolver";
20 import { exactIterator } from "./exact_iterator";
21
22 export const ftWordsTreeName = "words-tree";
23 export const ftOriginalField = "original";
24 export const ftExactField = "exact";
25
26 export function getFullTextFieldName(fieldType: string, field: string):
   ↪  string {
27   return "_" + fieldType + ":" + field;
28 }
29
30 export class ftIndexer extends exactIndexer {
```

```
  baseIndexer: exactIndexer;
  fullTextOptions: FullTextOptions;

  constructor(
    storage: Store,
    controller: Controller,
    exactOptions: ExactOptions,
    fullTextOptions: FullTextOptions
  ) {
    super(storage, exactOptions, controller);
    this.fullTextOptions = getDefaultFullTextOptions(fullTextOptions);
  }

  protected async newFullBatch(old?: exactMetaVersions): Promise<Batch> {
    let version = await this.generateNewVersionID();
    let allItems = new ItemSet();
    if (old && old.versions.length > 0) {
      allItems = new ItemSet(old.versions[0].allItems);
    }
    let items = new ItemSet();
    let ctx = new exactContext(version, {});
    return new ftBatch(
      ctx,
      allItems,
      items,
      old,
      this,
      this.storage,
      this.exactOptions,
      this.controller,
      this.fullTextOptions
    );
  }

  async rebuild(): Promise<Batch> {
    return this.newFullBatch();
  }

  async update(): Promise<Batch> {
```

```
    let old: exactMetaVersions = await this.storage.get(
      getMetaTable(this.exactOptions),
      getMetaKey(this.exactOptions)
    );
    if (!old) {
      return this.rebuild();
    }
    return this.newFullBatch(old);
  }

  newFullOptimizer(
    old: exactMetaVersions,
    sources: ContextGetter[],
    target: exactContext
  ): Optimizer {
    let allItems = new ItemSet(old.versions[0].allItems);
    return new ftOptimizer(
      old,
      target,
      allItems,
      this.storage,
      this.exactOptions,
      sources,
      this.controller
    );
  }

  async optimize() {
    let old: exactMetaVersions = await this.storage.get(
      getMetaTable(this.exactOptions),
      getMetaKey(this.exactOptions)
    );
    if (old && old.versions.length >= this.exactOptions.mergeMinVersions)
    ↪ {
      let sources = getSourcesVersion(old);
      let version = await this.generateNewVersionID();
      let target = new exactContext(version, {});
      let optimizer: Optimizer = this.newFullOptimizer(old, sources,
      ↪ target);
```

```
      await this.controller.optimize(sources, optimizer, target);
      await this.runGC();
    }
  }

  search(expression: Expression): Promise<IndexIterator> {
    return this.searchWithOptions(expression);
  }

  async searchWithOptions(
    expression: Expression,
    options?: SearchOptions
  ): Promise<IndexIterator> {
    let [all, meta] = await this.getResolverData(options.version);
    let data = new ftResolver(
      all,
      meta,
      this.controller,
      this.exactOptions,
      false,
      this.fullTextOptions
    );
    let result = await expression.evaluateBitmap(data);
    if (
      (result === null || result.isEmpty()) &&
      this.fullTextOptions.maxDistance > 0
    ) {
      data = new ftResolver(
        all,
        meta,
        this.controller,
        this.exactOptions,
        true,
        this.fullTextOptions
      );
      result = await expression.evaluateBitmap(data);
    }
    let [results, info] = this.getIterator(meta, result, options);
    return new exactIterator(results, this.controller, meta, info);
```

```
  }
}
```

Arquivo fulltext_batch.ts

```
import { exactBatch } from "./exact_batch";
import { MapKey, TreeNode, Controller } from "./base";
import { compareMapKey, getNodes } from "./utilities";
import { StringSetBuilder, StringSetNode } from "./stringset";
import {
  getFullTextFieldName,
  ftOriginalField,
  ftExactField,
  ftWordsTreeName,
  ftIndexer
} from "./fulltext";
import { FullTextOptions } from "./fulltext_options";
import { exactContext } from "./exact_context";
import { exactMetaVersions } from "./exact_models";
import { exactIndexer } from "./exact";
import { Store } from "../kv/base";
import { ExactOptions } from "./exact_options";
import { ItemSet } from "./itemset";

type ftBitmapChar = {
  field: string;
  value: string;
  bitmaps: Set<string>;
};

export default class ftBatch extends exactBatch {
  private fullText: ftIndexer;
  private fullTextOptions: FullTextOptions;

  constructor(
    ctx: exactContext,
    allItems: ItemSet,
    items: ItemSet,
    old: exactMetaVersions,
    fullText: ftIndexer,
```

```
    storage: Store,
    exactOptions: ExactOptions,
    controller: Controller,
    fullTextOptions: FullTextOptions
  ) {
    super(
      ctx,
      allItems,
      items,
      old,
      fullText,
      storage,
      exactOptions,
      controller
    );
    this.fullText = fullText;
    this.fullTextOptions = fullTextOptions;
  }

  addMapKey(char: ftBitmapChar) {
    if (char.field.length == 0 || char.value.length == 0) {
      return;
    }
    let newKey: MapKey = {
      field: char.field,
      value: char.value
    };
    let mapKey = this.keys.get(newKey);
    let result: ItemSet;
    if (this.keys.has(newKey) && !!mapKey) {
      let oldBitmap = this.bitmaps[mapKey.bitmap];
      oldBitmap.count--;
      result = oldBitmap.Bitmap;
      delete this.bitmaps[mapKey.bitmap];
      if (oldBitmap.count > 0) {
        result = oldBitmap.Bitmap.Clone();
        this.bitmaps[mapKey.bitmap] = oldBitmap;
      }
    } else {
```

```
      mapKey.index = this.keys.size;
      result = new ItemSet();
    }
    for (let bitmapKey of char.bitmaps.keys()) {
      if (bitmapKey != mapKey.bitmap) {
        result.or(this.bitmaps[bitmapKey].Bitmap);
      }
    }
    let resultEncoded = this.bitmapEncode(result);
    let bitmap = this.bitmaps[resultEncoded];
    if (!!bitmap) {
      result = bitmap.Bitmap;
    }
    bitmap.Bitmap = result;
    bitmap.Count++;
    this.bitmaps[resultEncoded] = bitmap;
    mapKey.bitmap = resultEncoded;
    this.keys.set(newKey, mapKey);
    for (let [key, value] of this.datametas.entries()) {
      if (result.get(value.counter) && !value.keys.get(mapKey.index)) {
        value.keys.set(mapKey.index, 1);
        this.datametas.set(key, value);
      }
    }
  }

  async commit() {
    let original = getFullTextFieldName(ftOriginalField, "");
    let originalLen = original.length;
    let keyList: MapKey[] = [];
    let maxLen: number;
    for (let key of this.keys.keys()) {
      if (key.field.indexOf(original) === 0) {
        keyList.push(key);
        if (key.value.length > maxLen) {
          maxLen = key.value.length;
        }
      }
    }
```

```
    let nodes: TreeNode[] = [];
    if (keyList.length > 0) {
      keyList.sort(compareMapKey);
      let tree = new StringSetBuilder();
      let seenValue: string;
      let seenNode: StringSetNode | null = null;

      if (this.fullTextOptions.maxPrefixLength < maxLen) {
        maxLen = this.fullTextOptions.maxPrefixLength;
      }
      let chars: ftBitmapChar[] = new Array(maxLen);
      for (let i = 0; i < maxLen; i++) {
        chars[i] = {
          field: "",
          value: "",
          bitmaps: new Set<string>()
        };
      }
      for (let key of keyList) {
        if (seenValue != key.field || seenNode === null) {
          seenValue = key.field;
          seenNode = tree.insertPrefix(null,
          ↪  seenValue.substr(originalLen));
          seenNode = tree.insertPrefix(seenNode, "|");
        }
        let value = key.value;
        let bitmap = this.keys.get(key).bitmap;

        for (let i = 0, l = Math.min(value.length, maxLen); i < maxLen;
        ↪  i++) {
          let char = chars[i];
          if (char.field != key.field || char.value != value.substr(0, i
          ↪  + 1)) {
            this.addMapKey(char);
            char.field = key.field;
            char.value = value.substr(0, i + 1);
            char.bitmaps.clear();
          }
          char.bitmaps.add(bitmap);
```

```
150           chars[i] = char;
151         }
152         tree.insertWord(seenNode, value);
153       }
154       tree.finish();
155       for (let char of chars) {
156         this.addMapKey(char);
157       }
158       nodes = getNodes(tree);
159     }
160     await this._commit(() => {
161       if (nodes.length > 0) {
162         this.inserter.commitTree(ftWordsTreeName, nodes);
163       }
164     });
165     let numVersions = this.old.versions.length + 1;
166     let autoMerge = this.exactOptions.autoMergeMinVersions;
167     if (autoMerge > 0 && autoMerge > numVersions) {
168       await this.fullText.optimize();
169     }
170   }
171
172   async update(values: { [key: string]: string }, itemKey: string, item:
      any) {
173     await this.remove(itemKey);
174     for (let [field, value] of Object.entries(values)) {
175       await this.setField(field, value, itemKey, item);
176     }
177   }
178
179   async set(value: string, itemKey: string, item: any) {
180     return this.setField(this.exactOptions.defaultField, value, itemKey,
        item);
181   }
182
183   async setField(field: string, value: string, itemKey: string, item:
      any) {
184     if (this.fullTextOptions.isExact(field)) {
185       let exactField = getFullTextFieldName(ftExactField, field);
```

```
      return super.setField(exactField, value, itemKey, item);
    }
    let originalField = getFullTextFieldName(ftOriginalField, field);
    let tokens = this.fullTextOptions.tokenizer(field, value);
    for (let token of tokens) {
      await super.setField(originalField, token, itemKey, item);
    }
  }
}
```

Arquivo fulltext_optimizer.ts

```
import { StringSetNode, StringSetBuilder } from "./stringset";
import exactOptimizer from "./exact_optimizer";
import { ContextGetter, Controller, Inserter } from "./base";
import { compareString, CompareResult, getNodes } from "./utilities";
import { ftWordsTreeName } from "./fulltext";
import { exactMetaVersions } from "./exact_models";
import { exactContext } from "./exact_context";
import { ItemSet } from "./itemset";
import { Store } from "../kv/base";
import { ExactOptions } from "./exact_options";

type ftTreeOptimizerQueueItem = {
  node: number;
  source: number;
  fullBuf: string;
  parent: StringSetNode | null;
};

function compareFullBuf(
  a: ftTreeOptimizerQueueItem,
  b: ftTreeOptimizerQueueItem
): CompareResult {
  return compareString(a.fullBuf, b.fullBuf);
}

export default class ftOptimizer extends exactOptimizer {
  private sources: ContextGetter[];
  private controller: Controller;
```

```
  private inserter: Inserter;

  constructor(
    old: exactMetaVersions,
    ctx: exactContext,
    allItems: ItemSet,
    storage: Store,
    options: ExactOptions,
    sources: ContextGetter[],
    controller: Controller
  ) {
    super(old, ctx, allItems, storage, options);
    this.sources = sources;
    this.controller = controller;
    this.inserter = controller.insert(ctx);
  }

  private async commitTree() {
    let queue: ftTreeOptimizerQueueItem[] = [];
    for (let sourceID = 0, l = this.sources.length; sourceID < l;
    ↪  sourceID++) {
      queue.push({
        source: sourceID,
        node: 0,
        fullBuf: "",
        parent: null
      });
    }
    queue.sort(compareFullBuf);
    let treeBuilder = new StringSetBuilder();
    while (queue.length > 0) {
      let item = queue.shift();
      let node = await this.controller.getTreeNode(
        this.sources[item.source],
        ftWordsTreeName,
        item.node
      );
      if (item.node == 0 && !node) {
        // It may happen that some versions don't have a tree, because
```

```
        // trees are incremental with each other and some versions may just
        // delete a bunch of things, so we need to ignore not found errors
        // while trying to load the root of these trees
        continue;
      }
      let parent = item.parent;
      let bufLen = item.fullBuf.length;
      let key: string = "";
      if (bufLen > 0) {
        key = item.fullBuf.substr(bufLen - 1);
      }
      if (node.final) {
        parent = treeBuilder.insertWord(parent, key);
      } else {
        parent = treeBuilder.insertPrefix(parent, key);
      }
      for (let edge of node.edges) {
        queue.push({
          source: item.source,
          node: edge.value,
          fullBuf: item.fullBuf + edge.key,
          parent: parent
        });
      }
      queue.sort(compareFullBuf);
    }
    treeBuilder.finish();
    let nodes = getNodes(treeBuilder);
    await this.inserter.commitTree(ftWordsTreeName, nodes);
  }

  async commit() {
    await this.commitTree();
    await super.commit();
  }
}
```

Arquivo fulltext_options.ts

```
export type FullTextOptions = {
  isExact?: (field: string) => boolean;
  tokenizer?: (field: string, value: string) => string[];
  maxUserDistance?: number;
  maxTreeDistance?: number;
  maxDistance?: number;
  maxPrefixLength?: number;
};

function DefaultIsExact(): boolean {
  return false;
}

const removePontuaction = /[^a-zA-Z0-9\\s]/g;
const removeAcents = /[\u0300-\u036f]/g;

function DefaultTokenizer(field: string, value: string): string[] {
  value = value.normalize("NFD");
  value = value.replace(removeAcents, "");
  value = value.replace(removePontuaction, "");
  return value.split(" ");
}

export function getDefaultFullTextOptions(
  options: FullTextOptions
): FullTextOptions {
  if (!options.isExact) {
    options.isExact = DefaultIsExact;
  }
  if (!options.tokenizer) {
    options.tokenizer = DefaultTokenizer;
  }
  return options;
}
```

Arquivo fulltext_resolver.ts

```
import { exactResolver } from "./exact_resolver";
```

```
import { exactIndexer } from "./exact";
import { ItemSet } from "./itemset";
import { TreeEdge, ContextGetter, TreeNode, Controller } from "./base";
import {
  ftWordsTreeName,
  getFullTextFieldName,
  ftExactField,
  ftOriginalField
} from "./fulltext";
import { FullTextOptions } from "./fulltext_options";
import { getContextVersion, exactMetaVersions } from "./exact_models";
import { ExactOptions } from "./exact_options";

type ftTreePrefixQueueItem = {
  node: number;
  // Full buffer of the edges navigated from the root to this node
  fullBuf: string;
};

type ftTreeSymSpellQueueItem = {
  node: number;
  // Number of edges "ignored" (not part of the full buffer) already
  bufDistance: number;
  // Full buffer of the edges navigated from the root to this node
  fullBuf: string;
  // Index of the next character to check on the user buffer
  userIndex: number;
  // Number of characters "ignored" on the user buffer
  userDistance: number;
};

export class ftResolver extends exactResolver {
  private enableCorrections: boolean;
  private ftOptions: FullTextOptions;

  constructor(
    all: ItemSet,
    meta: exactMetaVersions,
    controller: Controller,
```

```
    exactOptions: ExactOptions,
    enableCorrections: boolean,
    ftOptions: FullTextOptions
  ) {
    super(all, meta, controller, exactOptions);
    this.enableCorrections = enableCorrections;
    this.ftOptions = ftOptions;
  }

  getEdge(edges: TreeEdge[], value: string): number | undefined {
    let edgeResult = edges.find(edge => edge.key === value);
    if (!edgeResult) {
      // The root edge will never be referenced here, because the tree is
      ↪  acyclic.
      return;
    }
    return edgeResult.value;
  }

  async getPrefixes(
    ctx: ContextGetter,
    root: TreeNode,
    value: string,
    results: Set<string>
  ): Promise<Set<string>> {
    let controller = this.controller;
    let queue: ftTreePrefixQueueItem[] = [
      {
        node: root.id,
        fullBuf: value
      }
    ];
    while (queue.length > 0) {
      let elem = queue.shift();
      let node: TreeNode;
      if (elem.node == root.id) {
        node = root;
      } else {
```

```
78          node = await controller.getTreeNode(ctx, ftWordsTreeName,
            ↪  elem.node);
79        }
80        if (node.final) {
81          results.add(elem.fullBuf);
82        }
83        for (let edge of node.edges) {
84          queue.push({
85            node: edge.value,
86            fullBuf: elem.fullBuf + edge.key
87          });
88        }
89      }
90      return results;
91    }
92
93    async symspell(
94      ctx: ContextGetter,
95      root: TreeNode,
96      value: string,
97      results: Set<string>
98    ): Promise<Set<string>> {
99      let controller = this.controller;
100     let { maxTreeDistance, maxUserDistance, maxDistance } =
      ↪  this.ftOptions;
101     let queue: ftTreeSymSpellQueueItem[] = [
102       {
103         node: root.id,
104         bufDistance: 0,
105         fullBuf: "",
106         userDistance: 0,
107         userIndex: 0
108       }
109     ];
110     while (queue.length > 0) {
111       let elem = queue.shift();
112       let node: TreeNode;
113       if (elem.node == root.id) {
114         node = root;
```

```
115         } else {
116           node = await controller.getTreeNode(ctx, ftWordsTreeName,
              ↪ elem.node);
117         }
118         if (node.final && elem.userIndex == value.length) {
119           results.add(elem.fullBuf);
120         }
121         let actualChar: string;
122         // We need to check if userIndex is inside the string
123         // If it is, we cannot use actualChar to compare with edges and so
            ↪ on
124         // because we're already in the end of the string
125         let compareChar = value.length > elem.userIndex;
126         if (compareChar) {
127           actualChar = value[elem.userIndex];
128           let edge = this.getEdge(node.edges, actualChar);
129           if (
130             edge === 0 &&
131             elem.userDistance < maxUserDistance &&
132             elem.userDistance + elem.bufDistance + 1 < maxDistance
133           ) {
134             queue.push({
135               node: elem.node,
136               fullBuf: elem.fullBuf,
137               bufDistance: elem.bufDistance,
138               userIndex: elem.userIndex + 1,
139               userDistance: elem.userDistance + 1
140             });
141           }
142         }
143         for (let edge of node.edges) {
144           let nextBuf = elem.fullBuf + edge.key;
145           if (compareChar && actualChar == edge.key) {
146             // Found an edge with that letter
147             // Consume character: increment userIndex without increasing
                ↪ userDistance
148             queue.push({
149               node: edge.value,
150               fullBuf: nextBuf,
```

```
            bufDistance: elem.bufDistance,
            userIndex: elem.userIndex + 1,
            userDistance: elem.userDistance
          });
        } else if (
          elem.userDistance + elem.bufDistance + 1 < maxDistance &&
          elem.bufDistance < maxTreeDistance
        ) {
          // Ignore edge
          // Also doesn't change userIndex or userDistance
          queue.push({
            node: edge.value,
            fullBuf: nextBuf,
            bufDistance: elem.bufDistance + 1,
            userIndex: elem.userIndex,
            userDistance: elem.userDistance
          });
        }
      }
    }
    return results;
  }

  async traverse(
    ctx: ContextGetter,
    node: TreeNode,
    value: string
  ): Promise<boolean> {
    let controller = this.controller;
    let ok: boolean = true;
    let index: number = 0;
    while (ok && index < value.length) {
      let nextID = this.getEdge(node.edges, value[index]);
      ok = nextID > 0;
      if (ok) {
        node = await controller.getTreeNode(ctx, ftWordsTreeName,
        ↪  nextID);
        index++;
      }
```

```
    }
    return ok;
  }

  async prefixesVersion(
    ctx: ContextGetter,
    rootNode: string,
    value: string,
    result: Set<string>
  ): Promise<Set<string>> {
    let controller = this.controller;
    let node: TreeNode = await controller.getTreeNode(ctx,
    ↪  ftWordsTreeName, 0);
    // traverse will return false if the field cannot be found, so
    // at least if the field doesn't exist we will short-circuit
    // here
    let ok = await this.traverse(ctx, node, rootNode);
    return ok ? this.getPrefixes(ctx, node, value, result) : result;
  }

  async prefixes(
    field: string,
    value: string,
    uniqueify: Set<string>
  ): Promise<Set<string>> {
    // Fields have that suffix to split between fields and values:
    let fieldRunes = field + "|" + value;
    // Scan every version present in the tree?
    for (let version of this.meta.versions) {
      uniqueify = await this.prefixesVersion(
        getContextVersion(this.meta, version.version),
        fieldRunes,
        value,
        uniqueify
      );
    }
    return uniqueify;
  }

```

```
  async correctorVersion(
    ctx: ContextGetter,
    rootNode: string,
    value: string,
    result: Set<string>
  ): Promise<Set<string>> {
    let controller = this.controller;
    let node: TreeNode = await controller.getTreeNode(ctx,
    ↪  ftWordsTreeName, 0);
    // traverse will return false if the field cannot be found, so
    // at least if the field doesn't exist we willl short-circuit
    // here
    let ok = await this.traverse(ctx, node, rootNode);
    return ok ? this.symspell(ctx, node, value, result) : result;
  }

  async corrector(
    field: string,
    value: string,
    uniqueify: Set<string>
  ): Promise<Set<string>> {
    if (this.enableCorrections) {
      // Fields have that suffix to split between fields and values:
      let fieldRunes = field + "|";
      let valueRunes = value;
      // Scan every version present in the tree?
      for (let version of this.meta.versions) {
        uniqueify = await this.correctorVersion(
          getContextVersion(this.meta, version.version),
          fieldRunes,
          valueRunes,
          uniqueify
        );
      }
    }
    return uniqueify;
  }

  get(value: string): Promise<ItemSet> {
```

```
    return this.getField(this.exactOptions.defaultField, value);
  }

  async getField(field: string, value: string): Promise<ItemSet> {
    if (this.ftOptions.isExact(field)) {
      let exactField = getFullTextFieldName(ftExactField, field);
      return super.getField(exactField, value);
    }
    let tokens = this.ftOptions.tokenizer(field, value);
    let valueBitmap: ItemSet;
    let uniqueify = new Set<string>();
    let originalField = getFullTextFieldName(ftOriginalField, field);
    for (let token of tokens) {
      let mapKeyBitmap: ItemSet = await super.getField(originalField,
      ↪  token);
      if (
        token.length > this.ftOptions.maxPrefixLength ||
        mapKeyBitmap.isEmpty()
      ) {
        uniqueify = await this.prefixes(field, token, uniqueify);
        for (let prefix of uniqueify) {
          let prefixBitmap: ItemSet = await super.getField(
            originalField,
            prefix
          );
          if (prefixBitmap != null) {
            if (mapKeyBitmap === null) {
              mapKeyBitmap = prefixBitmap;
            } else {
              mapKeyBitmap = mapKeyBitmap.or(prefixBitmap);
            }
          }
        }
        uniqueify.clear();
      }
      if (
        this.enableCorrections &&
        this.ftOptions.maxDistance > 0 &&
        (mapKeyBitmap === null || mapKeyBitmap.isEmpty())
```

```
      ) {
        // Apply corrections
        uniqueify = await this.corrector(field, token, uniqueify);
        for (let correction of uniqueify) {
          var correctionBitmap: ItemSet = await super.getField(
            originalField,
            correction
          );
          if (correctionBitmap !== null) {
            if (mapKeyBitmap === null) {
              mapKeyBitmap = correctionBitmap;
            } else {
              mapKeyBitmap = mapKeyBitmap.or(correctionBitmap);
            }
          }
        }
        uniqueify.clear();
      }
      if (mapKeyBitmap !== null) {
        if (valueBitmap === null) {
          valueBitmap = mapKeyBitmap;
        } else {
          valueBitmap = valueBitmap.and(mapKeyBitmap);
        }
      }
    }
    if (valueBitmap !== null) {
      valueBitmap = valueBitmap.and(this.all);
    }
    return valueBitmap;
  }
}
```

Arquivo indexer.ts

```
export * from "./base";
export * from "./expression";
export * from "./exact";
export * from "./fulltext";
```

Arquivo itemset.ts

```
import { compareNumber } from "./utilities";

export class ItemSet {
  private values: number[];
  constructor(values: number[] = []) {
    this.values = values.slice();
    this.sort();
  }

  private sort() {
    this.values.sort(compareNumber);
  }

  add(item: number): boolean {
    if (this.values.includes(item)) {
      return false;
    }
    this.values.push(item);
    this.sort();
    return true;
  }

  remove(item: number): boolean {
    let index = this.values.indexOf(item);
    let found = index !== -1;
    if (found) {
      this.values.splice(index, 1);
    }
    return found;
  }

  lsb(): number {
    return Math.min(...this.values);
  }

  msb(): number {
    return Math.max(...this.values);
```

```
  }

  get(item: number): boolean {
    return this.values.includes(item);
  }

  clone(): ItemSet {
    return new ItemSet(this.values);
  }

  private baseAnd(other: ItemSet, intersect: boolean): ItemSet {
    return new ItemSet(
      this.values.filter(function(item) {
        return other.get(item) === intersect;
      })
    );
  }

  and(other: ItemSet): ItemSet {
    return this.baseAnd(other, true);
  }

  andNot(other: ItemSet): ItemSet {
    return this.baseAnd(other, false);
  }

  or(other: ItemSet): ItemSet {
    return new ItemSet(
      this.values.concat(
        other.values.filter(item => {
          return !this.get(item);
        })
      )
    );
  }

  setRange(start: number, end: number, add: 1 | 0) {
    if (add === 1) {
      for (let c = start; c < end; c++) {
```

```
        if (!this.values.includes(c)) {
          this.values.push(c);
        }
      }
      this.sort();
      return;
    }
    this.values = this.values.filter(function(item) {
      return item >= start && item < end;
    });
  }

  equals(other: ItemSet): boolean {
    let cardinality = this.cardinality();
    if (cardinality !== other.cardinality()) {
      return false;
    }
    for (let i = 0; i < cardinality; i++) {
      if (this.values[i] !== other.values[i]) {
        return false;
      }
    }
    return true;
  }

  clear() {
    this.values = [];
  }

  set(item: number, add: 1 | 0): boolean {
    if (add === 1) {
      return this.add(item);
    }
    return this.remove(item);
  }

  select(index: number) {
    return this.values[index];
  }
```

```
116
117   cardinality(): number {
118     return this.values.length;
119   }
120
121   isEmpty() {
122     return this.cardinality() === 0;
123   }
124
125   toArray(): number[] {
126     return this.values.slice();
127   }
128
129   toString(): string {
130     let s = "";
131     for (let value of this.values) {
132       s += value + "|";
133     }
134     return s;
135   }
136 }
```

Arquivo stringset.test.ts

```
1  import { StringSetBuilder, StringSetNode } from "./stringset";
2
3  function getStrings(tree: StringSetBuilder): string[] {
4    let result: string[] = [];
5    type item = {
6      fullBuf: string;
7      node: StringSetNode;
8    };
9    let queue: item[] = [
10     {
11       node: tree.root,
12       fullBuf: ""
13     }
14   ];
15   while (queue.length > 0) {
16     let elem = queue.pop();
```

```
    if (!elem) {
      break;
    }
    if (elem.node.final) {
      result.push(elem.fullBuf);
    }
    for (let [key, value] of Object.entries(elem.node.edges)) {
      queue.push({
        node: value,
        fullBuf: elem.fullBuf + key
      });
    }
  }
  return result.sort();
}

function countNodes(tree: StringSetBuilder): number {
  let found = new Set<number>();
  let queue: StringSetNode[] = [tree.root];
  while (queue.length > 0) {
    let elem = queue.pop();
    if (!elem) {
      break;
    }
    if (!found.has(elem.id)) {
      found.add(elem.id);
      for (let edge of Object.values(elem.edges)) {
        if (!found.has(edge.id)) {
          queue.push(edge);
        }
      }
    }
  }
  return found.size;
}

let builder = new StringSetBuilder();

let words: string[] = ["tap", "tap", "taps", "top", "tops"];
```

```
for (let word of words) {
  builder.insertWord(null, word);
}
builder.finish();

console.assert(getStrings(builder).join("|") ===
↪  words.slice(1).join("|"));
console.assert(countNodes(builder) === 6);
```

Arquivo stringset.ts

```
export type StringSetNode = {
  readonly edges: { [char: string]: StringSetNode };
  readonly id: number;
  readonly char: string;
  readonly lastChildKey: string;
  final: boolean;
};

function getNodeId(node: StringSetNode) {
  return node.id;
}

function hash(node: StringSetNode) {
  let edges = Object.values(node.edges).map(getNodeId);
  let suffix = edges.length > 0 ? "|" + edges.sort().join("|") : "";
  return node.char + "|" + (node.final ? "1" : "0") + suffix;
}

export class StringSetBuilder {
  public root: StringSetNode;
  private idCounter: number;
  private register: { [hash: string]: StringSetNode };

  constructor() {
    this.idCounter = 0;
    this.register = {};
    this.root = {
      id: this.idCounter++,
      char: "",
```

```
      lastChildKey: "",
      final: false,
      edges: {}
    };
  }
  public insertPrefix(lastState: StringSetNode | null = null, word:
  ↪ string) {
    let commonPrefixLength = 0;
    let size = word.length;
    lastState = lastState === null ? this.root : lastState;
    for (let char of word) {
      if (lastState.lastChildKey != char) {
        if (lastState.lastChildKey > char) {
          throw new Error(
            "Insertions must be performed in lexicographical order"
          );
        }
        break;
      }
      lastState = lastState.edges[word[commonPrefixLength]];
      commonPrefixLength++;
    }

    if (commonPrefixLength == size) {
      return;
    }

    this.replaceOrRegister(lastState);
    for (let i = commonPrefixLength, l = word.length; i < l; i++) {
      let char = word[i];
      let id = this.idCounter++;
      let newNode: StringSetNode = {
        id,
        char,
        lastChildKey: "",
        final: false,
        edges: {}
      };
      lastState.edges[char] = newNode;
```

```
      lastState = newNode;
    }
    return lastState;
  }

  public insertWord(lastState: StringSetNode, value: string):
  ↪  StringSetNode {
    let node = this.insertPrefix(lastState, value);
    node.final = true;
    return node;
  }

  private replaceOrRegister(node: StringSetNode) {
    if (Object.keys(node.edges).length == 0) {
      return;
    }
    let child = node.edges[node.lastChildKey];
    this.replaceOrRegister(child);
    let childHash = hash(child);
    let equiv = this.register[childHash];
    if (!!equiv) {
      node.edges[equiv.char] = equiv;
    } else {
      this.register[childHash] = child;
    }
  }

  public finish() {
    this.replaceOrRegister(this.root);
  }

  public getNodes(): Map<StringSetNode, number> {
    let mapping = new Map<StringSetNode, number>();
    let oldCount = mapping.size;
    mapping.set(this.root, oldCount);
    let newCount = mapping.size;
    while (oldCount < newCount) {
      oldCount = newCount;
      for (let node of mapping.keys()) {
```

```
        for (let edge of Object.values(node.edges)) {
          if (!mapping.has(edge)) {
            mapping.set(edge, mapping.size);
          }
        }
      }
      newCount = mapping.size;
    }
    return mapping;
  }
}

function getPrefixes(
  root: StringSetNode,
  value: string,
  results?: Set<string>
): Set<string> {
  if (results === undefined) {
    results = new Set();
  }
  type ftTreePrefixQueueItem = {
    node: StringSetNode;
    // Full buffer of the edges navigated from the root to this node
    fullBuf: string;
  };
  let queue: ftTreePrefixQueueItem[] = [
    {
      node: root,
      fullBuf: value
    }
  ];
  while (queue.length > 0) {
    let elem = queue.pop();
    if (!elem) {
      break;
    }
    let node = elem.node;
    // var node :  StringSetNode
    // if elem.node == root.id {
```

```
    //         node = root
    // } else {
    //         // node = controller.GetTreeNode(ctx, ftWordsTreeName,
    ↪  elem.node)
    // }
    if (node.final) {
      results.add(elem.fullBuf);
    }
    for (let edge of Object.values(node.edges)) {
      let nextBuf = elem.fullBuf + edge.char;
      queue.push({
        node: edge,
        fullBuf: nextBuf
      });
    }
  }
  return results;
}

type Options = {
  maxTreeDistance: number;
  maxUserDistance: number;
  maxDistance: number;
};

function symspell(
  options: Options,
  root: StringSetNode,
  value: string,
  results: Set<string>
): Set<string> {
  let maxTreeDistance = options.maxTreeDistance;
  let maxUserDistance = options.maxUserDistance;
  let maxDistance = options.maxDistance;

  type ftTreeSymSpellQueueItem = {
    node: StringSetNode;
    // Number of edges "ignored" (not part of the full buffer) already
    bufDistance: number;
```

```
    // Full buffer of the edges navigated from the root to this node
    fullBuf: string;
    // Index of the next character to check on the user buffer
    userIndex: number;
    // Number of characters "ignored" on the user buffer
    userDistance: number;
  };

  let queue: ftTreeSymSpellQueueItem[] = [
    {
      node: root,
      bufDistance: 0,
      fullBuf: "",
      userIndex: 0,
      userDistance: 0
    }
  ];
  while (queue.length > 0) {
    let elem = queue.pop();
    if (!elem) {
      break;
    }
    let { node } = elem;
    // var node TreeNode
    // if elem.node == uint32(root.ID) {
    //         node = root
    // } else {
    //         node, err = controller.GetTreeNode(ctx, ftWordsTreeName,
    ↪  elem.node)
    // }
    // if err == nil && node.Final && elem.userIndex == uint(len(value))
    ↪  {
    //         results[elem.fullBuf] = zero
    // }
    if (node.final && elem.userIndex === value.length) {
      results.add(elem.fullBuf);
    }
    let actualChar = value[elem.userIndex];
    // We need to check if userIndex is inside the string
```

```
    // If it is, we cannot use actualChar to compare with edges and so on
    // because we're already in the end of the string
    let compareChar = value.length > elem.userIndex;
    if (compareChar) {
      if (
        !node.edges[actualChar] &&
        elem.userDistance < maxUserDistance &&
        elem.userDistance + elem.bufDistance + 1 < maxDistance
      ) {
        let nextElem: ftTreeSymSpellQueueItem = {
          node,
          fullBuf: elem.fullBuf,
          bufDistance: elem.bufDistance,
          userIndex: elem.userIndex + 1,
          userDistance: elem.userDistance + 1
        };
        queue.push(nextElem);
      }
    }
    for (let edge of Object.values(node.edges)) {
      let nextBuf = elem.fullBuf + edge.char;
      if (compareChar && actualChar == edge.char) {
        // Found an edge with that letter
        // Consume character: increment userIndex without increasing
        ↪  userDistance
        queue.push({
          node: edge,
          fullBuf: nextBuf,
          bufDistance: elem.bufDistance,
          userIndex: elem.userIndex + 1,
          userDistance: elem.userDistance
        });
      } else if (
        elem.userDistance + elem.bufDistance + 1 < maxDistance &&
        elem.bufDistance < maxTreeDistance
      ) {
        // Ignore edge
        // Also doesn't change userIndex or userDistance
        queue.push({
```

```
            node: edge,
            fullBuf: nextBuf,
            bufDistance: elem.bufDistance + 1,
            userIndex: elem.userIndex,
            userDistance: elem.userDistance
          });
        }
      }
    }
    return results;
}
```

Arquivo utilities.ts

```
import { exactMetaVersions } from "./exact_models";
import { MapKey, TreeNode, TreeEdge } from "./base";
import { StringSetBuilder, StringSetNode } from "./stringset";
import { ItemSet } from "./itemset";

export function getItemsBitmaps({ versions }: exactMetaVersions):
↪  ItemSet[] {
    let result: ItemSet[] = [];
    for (let version of versions) {
      result.push(new ItemSet(version.items));
    }
    return result;
}

export type CompareResult = 0 | -1 | 1;

export function compareString(a: string, b: string): CompareResult {
    if (a === b) {
      return 0;
    }
    return a < b ? -1 : 1;
}

export function compareNumber(a: number, b: number): CompareResult {
    if (a === b) {
      return 0;
```

```
  }
  return a < b ? -1 : 1;
}

export function compareMapKey(a: MapKey, b: MapKey): CompareResult {
  if (a.field !== b.field) {
    return compareString(a.field, b.field);
  }
  return compareString(a.value, b.value);
}

function compareEdge(a: TreeEdge, b: TreeEdge): CompareResult {
  return compareString(a.key, b.key);
}

export function getNodes(tree: StringSetBuilder): TreeNode[] {
  let mapping = tree.getNodes();
  let nodes = new Array<TreeNode>(mapping.size);
  for (let [elem, index] of mapping.entries()) {
    nodes[index] = {
      id: index,
      final: elem.final,
      edges: []
    };
    let count: number = 0;
    for (let [key, edge] of Object.entries(elem.edges)) {
      nodes[index].edges[count] = {
        key: key,
        value: mapping.get(edge)
      };
      count++;
    }
    nodes[index].edges.sort(compareEdge);
  }
  return nodes;
}
```

B.2.31 Pasta src/js/service-worker/kv

Arquivo base.ts

```
export interface Batch {
  set(table: string, key: string, value: any): Promise<void>;
  commit(): Promise<void>;
}

export interface LimitedStore {
  get(table: string, key: string): Promise<any>;
  getMulti(table: string, keys: string[]): Promise<Map<string, any>>;
  set(table: string, key: string, value: any): Promise<void>;
  del(table: string, key: string): Promise<void>;
  batch(): Batch;
  generateID(table: string): Promise<string>;
}

export type StoreRow = { key: string; value: any };

export type Reference = {
  Table: string;
  Key: string;
};

export interface Store extends LimitedStore {
  getAll(table: string): Promise<StoreRow[]>;
  getAllKeys(table: string): Promise<string[]>;
  transaction(
    refs: string[],
    fn: (store: LimitedStore) => Promise<void>
  ): Promise<void>;
  close(): void;
}
```

Arquivo indexed.ts

```
import { Store, LimitedStore, Batch, StoreRow, Reference } from "./base";
import { IDBPDatabase, IDBPTransaction } from "idb";
import cuid from "cuid";

class IndexedBatch implements Batch {
  private parent: LimitedStore;

```

```
  constructor(parent: LimitedStore) {
    this.parent = parent;
  }

  async set(table: string, id: string, value: any): Promise<void> {
    await this.parent.set(table, id, value);
  }

  commit() {
    return Promise.resolve();
  }
}

class IndexedTransaction implements LimitedStore {
  private tx: IDBPTransaction;

  constructor(tx: IDBPTransaction) {
    this.tx = tx;
  }

  async getMulti(table: string, keys: string[]): Promise<Map<string,
  ↪  any>> {
    let result = new Map<string, any>();
    for (let key of keys) {
      let value = await this.tx.objectStore(table).get(key);
      if (value) {
        result.set(key, value);
      }
    }
    return result;
  }

  async get(table: string, key: string): Promise<any> {
    let result = await this.tx.objectStore(table).get(key);
    if (!result) {
      throw new Error(`Key "${key}" not found in table "${table}`);
    }
    return result;
  }
```

```
46
47    async del(table: string, id: string): Promise<void> {
48      await this.tx.objectStore(table).delete(id);
49    }
50
51    async set(table: string, id: string, value: any): Promise<void> {
52      await this.tx.objectStore(table).put(value, id);
53    }
54
55    async generateID(): Promise<string> {
56      return cuid();
57    }
58
59    batch() {
60      return new IndexedBatch(this);
61    }
62  }
63
64  export class IndexedStore implements Store {
65    private db: IDBPDatabase;
66    constructor(db: IDBPDatabase) {
67      this.db = db;
68    }
69
70    async get(table: string, key: string): Promise<any> {
71      let originalTx = this.db.transaction([table], "readonly");
72      let tx = new IndexedTransaction(originalTx);
73      let result = tx.get(table, key);
74      await originalTx.done;
75      return result;
76    }
77
78    async getMulti(table: string, keys: string[]): Promise<Map<string,
      ↪  any>> {
79      let originalTx = this.db.transaction([table], "readonly");
80      let tx = new IndexedTransaction(originalTx);
81      let result = tx.getMulti(table, keys);
82      await originalTx.done;
83      return result;
```

```
  }

  async set(table: string, key: string, value: any): Promise<void> {
    let originalTx = this.db.transaction([table], "readwrite");
    let tx = new IndexedTransaction(originalTx);
    tx.set(table, key, value);
    await originalTx.done;
  }

  async del(table: string, key: string): Promise<void> {
    let originalTx = this.db.transaction([table], "readwrite");
    let tx = new IndexedTransaction(originalTx);
    tx.del(table, key);
    await originalTx.done;
  }

  async transaction(
    tables: string[],
    fn: (store: LimitedStore) => void
  ): Promise<void> {
    let originalTx = this.db.transaction(tables, "readwrite");
    let tx = new IndexedTransaction(originalTx);
    try {
      fn(tx);
    } catch (e) {
      originalTx.abort();
      throw e;
    } finally {
      await originalTx.done;
    }
  }

  batch() {
    return new IndexedBatch(this);
  }

  async generateID(): Promise<string> {
    return cuid();
  }
```

```
123
124   async getAll(table: string): Promise<StoreRow[]> {
125     let result: StoreRow[] = [];
126     let transaction = this.db.transaction(table, "readonly");
127     let cursor = await transaction.store.openCursor();
128     while (cursor) {
129       result.push({ key: cursor.key.toString(), value: cursor.value });
130       cursor = await cursor.continue();
131     }
132     await transaction.done;
133     return result;
134   }
135
136   async getAllKeys(table: string): Promise<string[]> {
137     let result: string[] = [];
138     let transaction = this.db.transaction(table, "readonly");
139     let cursor = await transaction.store.openCursor();
140     while (cursor) {
141       result.push(cursor.key.toString());
142       cursor = await cursor.continue();
143     }
144     await transaction.done;
145     return result;
146   }
147
148   async close() {
149     return this.db.close();
150   }
151 }
```

B.2.32 Pasta src/js/service-worker/resolvers

Arquivo campus.ts

```
1 import { Database, CampusDefinition, ForeignKey } from "../database";
2 import Period, { getPeriod } from "./period";
3 import University, { getUniversity } from "./university";
4 import DisciplineOffer, { getDisciplineOffer } from "./discipline-offer";
5
6 export const TableCampus = "campus";
```

```
7
8  type CampusResolver = CampusDefinition["resolver"];
9  export default class Campus implements CampusResolver {
10   private db: Database;
11   public id: string;
12   public version: string;
13   private _period: string;
14   public name: string;
15   private universityID: string;
16   private _disciplineOffers: string[];
17
18   constructor(db: Database, data: CampusDefinition["schema"]) {
19     this.db = db;
20     this.id = data.id;
21     this.version = data.version;
22     this.name = data.name;
23     this._period = data.period;
24     this.universityID = data.universityID;
25     this._disciplineOffers = data.disciplineOffers;
26   }
27
28   period(): Promise<Period> {
29     return getPeriod(this.db, this._period);
30   }
31
32   university(): Promise<University> {
33     return getUniversity(this.db, this.universityID);
34   }
35
36   getDisciplineOffer(id: string): Promise<DisciplineOffer> {
37     return getDisciplineOffer(this.db, id);
38   }
39
40   disciplineOffers(): Promise<DisciplineOffer[]> {
41     return Promise.all(
42       this._disciplineOffers.map(this.getDisciplineOffer, this)
43     );
44   }
45
```

```
46    public async searchDisciplineOffers() {}
47    public async searchTeams() {}
48  }
49
50  export async function getCampus(db: Database, id: string):
    ↪  Promise<Campus> {
51    let row = await db.get(TableCampus, id);
52    return new Campus(db, row);
53  }
```

Arquivo course.ts

```
1   import { Database, CourseDefinition } from "../database";
2   import University, { getUniversity } from "./university";
3   import Habilitation, { getHabilitation } from "./habilitation";
4
5   export const TableCourse = "course";
6
7   type CourseResolvers = CourseDefinition["resolver"];
8
9   export default class Course implements CourseResolvers {
10    private db: Database;
11    public id: string;
12    public name: string;
13    public courseVersion: string;
14    public version: string;
15    public url?: string;
16    private universityID: string;
17    private _habilitations: string[];
18
19    constructor(db: Database, data: CourseDefinition["schema"]) {
20      this.db = db;
21      this.id = data.id;
22      this.version = data.version;
23      this.universityID = data.universityID;
24      this.name = data.name;
25      this.courseVersion = data.courseVersion;
26      this.url = data.url;
27      this._habilitations = data.habilitations;
28    }
```

```
29
30    university(): Promise<University> {
31      return getUniversity(this.db, this.universityID);
32    }
33
34    private getHabilitation(habilitation: string): Promise<Habilitation> {
35      return getHabilitation(this.db, habilitation);
36    }
37
38    habilitations(): Promise<Habilitation[]> {
39      return Promise.all(this._habilitations.map(this.getHabilitation,
        ↪  this));
40    }
41  }
42
43  export async function getCourse(db: Database, id: string):
    ↪  Promise<Course> {
44    let row = await db.get(TableCourse, id);
45    return new Course(db, row);
46  }
```

Arquivo discipline-offer.ts

```
1   import { SearchRequest, SearchResult } from "./search";
2   import { Database, DisciplineOfferDefinition, ForeignKey } from
    ↪  "../database";
3   import University, { getUniversity } from "./university";
4   import Period, { getPeriod } from "./period";
5   import Campus, { getCampus } from "./campus";
6   import Team, { getTeam } from "./team";
7
8   export const TableDisciplineOffer = "discipline_offer";
9
10  type DisciplineOfferResolvers = DisciplineOfferDefinition["resolver"];
11
12  export default class DisciplineOffer implements DisciplineOfferResolvers
    ↪  {
13    private db: Database;
14    public id: string;
15    public version: string;
```

```
16    private universityID: string;
17    public genericID: string;
18    public code: string;
19    public name: string;
20    public _period: string;
21    public _campus: string;
22    public _teams: string[];
23
24    constructor(db: Database, data: DisciplineOfferDefinition["schema"]) {
25      this.db = db;
26      this.id = data.id;
27      this.version = data.version;
28      this.universityID = data.universityID;
29      this._period = data.period;
30      this._campus = data.campus;
31      this.genericID = data.genericID;
32      this.code = data.code;
33      this.name = data.name;
34      this._teams = data.teams;
35    }
36
37    university(): Promise<University> {
38      return getUniversity(this.db, this.universityID);
39    }
40
41    period(): Promise<Period> {
42      return getPeriod(this.db, this._period);
43    }
44
45    campus(): Promise<Campus> {
46      return getCampus(this.db, this._campus);
47      University;
48    }
49
50    private getTeam(team: string): Promise<Team> {
51      return getTeam(this.db, team);
52    }
53
54    teams(): Promise<Team[]> {
```

```
    return Promise.all(this._teams.map(this.getTeam, this));
  }

  async searchTeams(args: SearchRequest): Promise<SearchResult<Team>> {
    return {
      results: [],
      info: {
        page: {
          after: 0,
          before: 0
        },
        query: {
          count: 0,
          last: 0,
          first: 0,
          version: ""
        }
      }
    };
  }
}

export async function getDisciplineOffer(
  db: Database,
  id: string
): Promise<DisciplineOffer> {
  let row = await db.get(TableDisciplineOffer, id);
  return new DisciplineOffer(db, row);
}
```

Arquivo discipline.ts

```
import { SearchRequest } from "./search";
import {
  RemoteDisciplineRelated,
  RemoteTable,
  RemoteTeamSearchResult,
  RemoteDisciplineOfferSearchResult,
  RemoteDisciplineRelatedItemType
} from "../../graphql";
```

```
import { Database, DisciplineDefinition, ForeignKey } from "../database";
import University, { getUniversity } from "./university";
import { getCourse } from "./course";
import { getHabilitation } from "./habilitation";

export const TableDiscipline = "discipline";

type SearchTeamsRequest = SearchRequest & {
  periodID: string;
};

type DisciplineResolvers = DisciplineDefinition["resolver"];

export default class Discipline implements DisciplineResolvers {
  private db: Database;
  public id: string;
  public version: string;
  private universityID: string;
  private _course: string;
  private _habilitation: string;
  public step: string;
  public genericID: string;
  public code: string;
  public name: string;
  public description: string;
  public type: string;
  public tables: RemoteTable[];
  public related: RemoteDisciplineRelated[];

  constructor(db: Database, data: DisciplineDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.version = data.version;
    this.universityID = data.universityID;
    this._course = data.course;
    this._habilitation = data.habilitation;
    this.step = data.step;
    this.genericID = data.genericID;
    this.code = data.code;
```

```
    this.name = data.name;
    this.description = data.description;
    this.type = data.type;
    this.tables = data.tables;
    this.related = data.related;
  }

  university(): Promise<University> {
    return getUniversity(this.db, this.universityID);
  }

  course() {
    return getCourse(this.db, this._course);
  }

  habilitation() {
    return getHabilitation(this.db, this._habilitation);
  }

  async relatedDisciplines(): Promise<Discipline[]> {
    let disciplineIds: Set<string> = new Set();
    let disciplineGenericIds: Set<string> = new Set();
    for (let relatedItems of this.related) {
      for (let item of relatedItems.items) {
        switch (item.type) {
          case RemoteDisciplineRelatedItemType.DisciplineGenericId:
            disciplineGenericIds.add(item.value);
            break;
          case RemoteDisciplineRelatedItemType.DisciplineId:
            disciplineIds.add(item.value);
            break;
        }
      }
    }
    // TODO: manage disciplineGenericIds
    let results: Promise<Discipline>[] = [];
    for (let disciplineId of disciplineIds) {
      results.push(getDiscipline(this.db, disciplineId));
    }
```

```
87      return Promise.all(results);
88    }
89
90    searchTeams(request: SearchTeamsRequest): RemoteTeamSearchResult {
91      return {
92        results: [],
93        info: {
94          page: {
95            after: 0,
96            before: 0
97          },
98          query: {
99            count: 0,
100           first: 0,
101           last: 0,
102           version: "hm"
103         }
104       }
105     };
106   }
107   searchDisciplineOffers(): RemoteDisciplineOfferSearchResult {
108     return {
109       results: [],
110       info: {
111         page: {
112           after: 0,
113           before: 0
114         },
115         query: {
116           count: 0,
117           first: 0,
118           last: 0,
119           version: "hm"
120         }
121       }
122     };
123   }
124 }
125
```

```
126 export async function getDiscipline(
127   db: Database,
128   id: string
129 ): Promise<Discipline> {
130   let row = await db.get(TableDiscipline, id);
131   return new Discipline(db, row);
132 }
```

Arquivo habilitation.ts

```
1  import { Database, HabilitationDefinition } from "../database";
2  import University, { getUniversity } from "./university";
3  import { RemoteTable, RemoteHabilitationLimit } from "../../graphql";
4  import Course, { getCourse } from "./course";
5  import Step, { getStep } from "./step";
6
7  export const TableHabilitation = "habilitation";
8
9  type HabilitationResolvers = HabilitationDefinition["resolver"];
10
11 export default class Habilitation implements HabilitationResolvers {
12   private db: Database;
13   public id: string;
14   public name: string;
15   public version: string;
16   private universityID: string;
17   private _course: string;
18   private _steps: string[];
19   tables: RemoteTable[];
20   limits: RemoteHabilitationLimit[];
21
22   constructor(db: Database, data: HabilitationDefinition["schema"]) {
23     this.db = db;
24     this.id = data.id;
25     this.version = data.version;
26     this.universityID = data.universityID;
27     this.name = data.name;
28     this._course = data.course;
29     this._steps = data.steps;
30     this.tables = data.tables;
```

```
    this.limits = data.limits;
  }

  university(): Promise<University> {
    return getUniversity(this.db, this.universityID);
  }

  course(): Promise<Course> {
    return getCourse(this.db, this._course);
  }

  private getStep(step: string): Promise<Step> {
    return getStep(this.db, step);
  }

  steps(): Promise<Step[]> {
    return Promise.all(this._steps.map(this.getStep, this));
  }
}

export async function getHabilitation(
  db: Database,
  id: string
): Promise<Habilitation> {
  let row = await db.get(TableHabilitation, id);
  return new Habilitation(db, row);
}
```

Arquivo index.ts

```
import { RemoteOfflineResponse, RemoteOfflineRequest } from
↪ "../../graphql";
import { Database } from "../database";
import Discipline, { getDiscipline } from "./discipline";
import University, { getUniversity, getAllUniversities } from
↪ "./university";
import Course, { getCourse } from "./course";
import Period, { getPeriod } from "./period";
import Campus, { getCampus } from "./campus";
import Habilitation, { getHabilitation } from "./habilitation";
```

```
import DisciplineOffer, { getDisciplineOffer } from "./discipline-offer";
import { getStep } from "./step";
import Plan, { getAllPlans, getPlan } from "./plan";

type OfflineRequest = {
  request: RemoteOfflineRequest;
};

type idRequest = {
  id: string;
};

export default class QueryResolver {
  private db: Database;

  constructor(db: Database) {
    this.db = db;
  }

  university({ id }: idRequest): Promise<University> {
    return getUniversity(this.db, id);
  }

  async universities() {
    return getAllUniversities(this.db);
  }

  course({ id }: idRequest): Promise<Course> {
    return getCourse(this.db, id);
  }

  async courses({ universityId }): Promise<Course[]> {
    let university = await getUniversity(this.db, universityId);
    return university.courses();
  }

  period({ id }: idRequest): Promise<Period> {
    return getPeriod(this.db, id);
  }
```

```
48
49    async periods({ universityId }): Promise<Period[]> {
50      let university = await getUniversity(this.db, universityId);
51      return university.periods();
52    }
53
54    campus({ id }: idRequest): Promise<Campus> {
55      return getCampus(this.db, id);
56    }
57
58    async campi({ periodID }): Promise<Campus[]> {
59      let period = await getPeriod(this.db, periodID);
60      return period.campi();
61    }
62
63    habilitation({ id }: idRequest): Promise<Habilitation> {
64      return getHabilitation(this.db, id);
65    }
66
67    async habilitations({ courseID }): Promise<Habilitation[]> {
68      let course = await getCourse(this.db, courseID);
69      return course.habilitations();
70    }
71
72    disciplineOffer({ id }: idRequest): Promise<DisciplineOffer> {
73      return getDisciplineOffer(this.db, id);
74    }
75
76    async disciplineOffers({ campusId }): Promise<DisciplineOffer[]> {
77      let campus = await getCampus(this.db, campusId);
78      return campus.disciplineOffers();
79    }
80
81    discipline({ id }: idRequest): Promise<Discipline> {
82      return getDiscipline(this.db, id);
83    }
84
85    async disciplines({ habilitationID, stepID }): Promise<Discipline[]> {
86      let habilitation = await getHabilitation(this.db, habilitationID);
```

```
    let steps = await habilitation.steps();
    for (let step of steps) {
      if (step.id === stepID) {
        return step.disciplines();
      }
    }
    return [];
  }

  plans(): Promise<Plan[]> {
    return getAllPlans(this.db);
  }

  plan({ id }: idRequest): Promise<Plan> {
    return getPlan(this.db, id);
  }

  async loggedUser() {}

  getOfflineUpdates({ request }: OfflineRequest): RemoteOfflineResponse {
    return {
      disciplines: [],
      disciplineOffers: [],
      disciplineOffersToDelete: [],
      disciplinesToDelete: [],
      habilitations: [],
      habilitationsToDelete: []
    };
  }
}
```

Arquivo period.ts

```
import { Database, PeriodDefinition, ForeignKey } from "../database";
import University, { getUniversity } from "./university";
import Campus, { getCampus } from "./campus";

export const TablePeriod = "period";

type PeriodResolver = PeriodDefinition["resolver"];
```

```

export default class Period implements PeriodResolver {
  private db: Database;
  public id: string;
  public name: string;
  public version: string;
  private universityID: string;
  private _campi: string[];

  constructor(db: Database, data: PeriodDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.name = data.name;
    this.version = data.version;
    this.universityID = data.universityID;
    this._campi = data.campi;
  }

  university(): Promise<University> {
    return getUniversity(this.db, this.universityID);
  }

  private getCampus(campus: string): Promise<Campus> {
    return getCampus(this.db, campus);
  }

  campi(): Promise<Campus[]> {
    return Promise.all(this._campi.map(this.getCampus, this));
  }
}

export async function getPeriod(db: Database, id: string):
↪  Promise<Period> {
  let row = await db.get(TablePeriod, id);
  return new Period(db, row);
}
```

Arquivo plan-version.ts

```
import { Database, PlanVersionDefinition } from "../database";
```

```
import {
  RemotePlanDisciplineOffer,
  RemotePlanPossibility,
  RemotePlanDiscipline
} from "../../graphql";

export const TablePlanVersion = "plan_version";

type PlanVersionResolver = PlanVersionDefinition["resolver"];

export default class PlanVersion implements PlanVersionResolver {
  private db: Database;
  public id: string;
  public disciplineOffers: RemotePlanDisciplineOffer[];
  public disciplines: RemotePlanDiscipline[];
  public possibilities: RemotePlanPossibility[];
  public savedAt: Date;
  public selectedPossibility: number;
  public totalPossibilities: number;

  constructor(db: Database, data: PlanVersionDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.disciplineOffers = data.disciplineOffers;
    this.disciplines = data.disciplines;
    this.possibilities = data.possibilities;
    this.savedAt = data.savedAt;
    this.selectedPossibility = data.selectedPossibility;
    this.totalPossibilities = data.totalPossibilities;
  }
}

export async function getPlanVersion(
  db: Database,
  id: string
): Promise<PlanVersion> {
  let row = await db.get(TablePlanVersion, id);
  return new PlanVersion(db, row);
}
```

Arquivo plan.ts

```ts
import { Database, PlanDefinition, ForeignKey } from "../database";
import University, { getUniversity } from "./university";
import Campus, { getCampus } from "./campus";
import PlanVersion, { getPlanVersion } from "./plan-version";

export const TablePlan = "plan";

type PlanResolver = PlanDefinition["resolver"];

export default class Plan implements PlanResolver {
  private db: Database;
  public id: string;
  public name: string;
  public lastModified: Date;
  public createdAt: Date;
  public public: boolean;
  private universityID: string;
  private _versions: string[];

  constructor(db: Database, data: PlanDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.name = data.name;
    this.universityID = data.universityID;
    this.lastModified = data.lastModified;
    this.createdAt = data.createdAt;
    this.public = data.public;
    this._versions = data.versions;
  }

  university(): Promise<University> {
    return getUniversity(this.db, this.universityID);
  }

  private getPlanVersion(version: string): Promise<PlanVersion> {
    return getPlanVersion(this.db, version);
  }
```

```

  version({ index }: { index: number }): Promise<PlanVersion> {
    let version = this._versions[index];
    if (!version) {
      throw new Error("Version not found");
    }
    return this.getPlanVersion(version);
  }

  versions(): Promise<PlanVersion[]> {
    return Promise.all(this._versions.map(this.getPlanVersion, this));
  }
}

export async function getPlan(db: Database, id: string): Promise<Plan> {
  let row = await db.get(TablePlan, id);
  return new Plan(db, row);
}

export async function getAllPlans(db: Database): Promise<Plan[]> {
  let rows = await db.getAll(TablePlan);
  return rows.map(row => new Plan(db, row));
}
```

Arquivo room.ts

```
import Period from "./period";
import University from "./university";
import Campus from "./campus";
import { RemoteRoom } from "../../graphql";
import Team from "./team";

type RoomSchema = Omit<
  RemoteRoom,
  "campus" | "period" | "university" | "searchTeams"
>;
export class Room implements RoomSchema {
  public id: string;
  public name: string;
  public description: string | undefined;
```

```
15    public olcode: string | undefined;
16    public genericID: string;
17    private team: Team;
18
19    constructor(team: Team, data: RoomSchema) {
20      this.id = data.id;
21      this.genericID = data.genericID;
22      this.name = data.name;
23      this.description = data.description;
24      this.olcode = data.olcode;
25      this.team = team;
26    }
27
28    period(): Promise<Period> {
29      return this.team.period();
30    }
31
32    campus(): Promise<Campus> {
33      return this.team.campus();
34    }
35
36    university(): Promise<University> {
37      return this.team.university();
38    }
39
40    searchTeams() {}
41  }
```

Arquivo schedule.ts

```
1  import { Database } from "../database";
2  import { RemoteHourMinute, RemoteSchedule } from "../../graphql";
3  import { Room } from "./room";
4  import Team from "./team";
5
6  export class Schedule implements Omit<RemoteSchedule, "room"> {
7    public start: RemoteHourMinute;
8    public end: RemoteHourMinute;
9    public dayOfWeek: number;
10   public room?: Room;
```

```
11
12    constructor(team: Team, data: RemoteSchedule) {
13      this.start = data.start;
14      this.end = data.end;
15      this.dayOfWeek = data.dayOfWeek;
16      if (data.room) {
17        this.room = new Room(team, data.room);
18      }
19    }
20  }
```

Arquivo search.ts

```
1   import {
2     RemoteSearchOptions,
3     RemoteSearchExpression,
4     RemoteSearchResult,
5     RemoteSearchOperator
6   } from "../../graphql";
7   import { Query } from "lunr";
8
9   export type SearchRequest<T = {}> = {
10    query?: RemoteSearchExpression;
11    options?: RemoteSearchOptions;
12  } & T;
13
14  export type SearchResult<T> = RemoteSearchResult & {
15    results: T[];
16  };
17
18  function checkValue({ value }: RemoteSearchExpression) {
19    let result = value !== undefined;
20    if (!result) {
21      throw new Error("Expression should have a value");
22    }
23    return result;
24  }
25
26  function checkAttribute({ attribute }: RemoteSearchExpression) {
27    let result = attribute !== undefined;
```

```
28    if (!result) {
29      throw new Error("Expression should have a attribute");
30    }
31    return result;
32  }
33
34  function checkExpressions(
35    { expressions }: RemoteSearchExpression,
36    size?: number
37  ): boolean {
38    if (expressions === undefined) {
39      throw new Error("Expression should have a list of expressions");
40    } else if (size !== undefined && expressions.length !== size) {
41      throw new Error(`Expressions list should have exactly ${size}
      ↪  elements`);
42    }
43    return expressions !== undefined;
44  }
45
46  function applyToQuery(query: Query, expression?: RemoteSearchExpression)
    ↪  {
47    if (!expression) {
48      expression = {
49        type: RemoteSearchOperator.All
50      };
51    }
52    switch (expression.type) {
53      case RemoteSearchOperator.FieldTerminal:
54        if (checkValue(expression) && checkAttribute(expression)) {
55        }
56        break;
57      case RemoteSearchOperator.Terminal:
58        if (checkValue(expression)) {
59        }
60        break;
61      case RemoteSearchOperator.Or:
62        if (checkExpressions(expression)) {
63        }
64        break;
```

```
    case RemoteSearchOperator.And:
      if (checkExpressions(expression)) {
      }
      break;
    case RemoteSearchOperator.Not:
      if (checkExpressions(expression, 1)) {
      }
      break;
  }
}
```

Arquivo step.ts

```
import { Database, StepDefinition } from "../database";
import Discipline, { getDiscipline } from "./discipline";

export const TableStep = "step";

type StepResolvers = StepDefinition["resolver"];

export default class Step implements StepResolvers {
  private db: Database;
  public id: string;
  public name: string;
  private _disciplines: string[];

  constructor(db: Database, data: StepDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.name = data.name;
    this._disciplines = data.disciplines;
  }

  private getDiscipline(discipline: string): Promise<Discipline> {
    return getDiscipline(this.db, discipline);
  }

  disciplines(): Promise<Discipline[]> {
    return Promise.all(this._disciplines.map(this.getDiscipline, this));
  }
```

```
}

export async function getStep(db: Database, id: string): Promise<Step> {
  let row = await db.get(TableStep, id);
  return new Step(db, row);
}
```

Arquivo teacher.ts

```
import { RemoteTeacher } from "../../graphql";
import Team from "./team";
import University from "./university";
import Period from "./period";

export type TeacherType = Omit<
  RemoteTeacher,
  "university" | "period" | "searchTeams"
>;

export class Teacher implements TeacherType {
  private team: Team;
  public id: string;
  public genericID: string;
  public name: string;
  public url?: string;

  constructor(team: Team, teacher: TeacherType) {
    this.team = team;
    this.id = teacher.id;
    this.genericID = teacher.genericID;
    this.name = teacher.name;
    this.url = teacher.url;
  }

  public university(): Promise<University> {
    return this.team.university();
  }

  public period(): Promise<Period> {
    return this.team.period();
```

```
  }
}
```

Arquivo team.ts

```
import { SearchRequest, SearchResult } from "./search";
import { Database, TeamDefinition } from "../database";
import University, { getUniversity } from "./university";
import Period, { getPeriod } from "./period";
import Campus, { getCampus } from "./campus";
import { RemoteVacancy, RemoteSchedule, RemoteTeacher } from
↪  "../../graphql";
import { Schedule } from "./schedule";
import { Teacher } from "./teacher";

export const TableTeam = "team";

type SearchTeamsRequest = SearchRequest & {
  periodID: string;
};

type TeamResolvers = TeamDefinition["resolver"];

export default class Team implements TeamResolvers {
  private db: Database;
  public id: string;
  public version: string;
  private universityID: string;
  public code: string;
  public vacancies?: RemoteVacancy;
  public _period: string;
  public _campus: string;
  private _teachers: RemoteTeacher[];
  private _schedules: RemoteSchedule[];

  constructor(db: Database, data: TeamDefinition["schema"]) {
    this.db = db;
    this.id = data.id;
    this.version = data.version;
    this.universityID = data.universityID;
```

```
35      this._period = data.period;
36      this._campus = data.campus;
37      this.code = data.code;
38      this.vacancies = data.vacancies;
39      this._teachers = data.teachers;
40      this._schedules = data.schedules;
41    }
42
43    university(): Promise<University> {
44      return getUniversity(this.db, this.universityID);
45    }
46
47    period(): Promise<Period> {
48      return getPeriod(this.db, this._period);
49    }
50
51    campus(): Promise<Campus> {
52      return getCampus(this.db, this._campus);
53    }
54
55    private getTeacher(teacher: RemoteTeacher) {
56      return new Teacher(this, teacher);
57    }
58
59    async teachers(): Promise<Teacher[]> {
60      return this._teachers.map(this.getTeacher);
61    }
62
63    private getSchedule(schedule: RemoteSchedule) {
64      return new Schedule(this, schedule);
65    }
66
67    async schedules(): Promise<Schedule[]> {
68      return this._schedules.map(this.getSchedule, this);
69    }
70
71    async searchTeams(args: SearchRequest): Promise<SearchResult<Team>> {
72      return {
73        results: [],
```

```
74        info: {
75          page: {
76            after: 0,
77            before: 0
78          },
79          query: {
80            count: 0,
81            last: 0,
82            first: 0,
83            version: ""
84          }
85        }
86      };
87    }
88  }
89
90  export async function getTeam(db: Database, id: string): Promise<Team> {
91    let row = await db.get(TableTeam, id);
92    return new Team(db, row);
93  }
```

Arquivo university.ts

```
1  import { UniversityDefinition, Database, ForeignKey } from "../database";
2  import { RemoteUniversityData } from "../../graphql";
3  import Period, { getPeriod } from "./period";
4  import Course, { getCourse } from "./course";
5
6  export const TableUniversity = "university";
7
8  type UniversityResolver = UniversityDefinition["resolver"];
9  export default class University implements UniversityResolver {
10   private db: Database;
11   public id: string;
12   public acronym: string;
13   public name: string;
14   public public: boolean;
15   public offers: RemoteUniversityData;
16   public habilitations: RemoteUniversityData;
17   private _courses: string[];
```

```
18    private _periods: string[];
19
20    constructor(db: Database, data: UniversityDefinition["schema"]) {
21      this.db = db;
22      this.id = data.id;
23      this.acronym = data.acronym;
24      this.name = data.name;
25      this.public = data.public;
26      this.offers = data.offers;
27      this.habilitations = data.habilitations;
28      this._courses = data.courses;
29      this._periods = data.periods;
30    }
31
32    private getCourse(course: string): Promise<Course> {
33      return getCourse(this.db, course);
34    }
35
36    courses(): Promise<Course[]> {
37      return Promise.all(this._courses.map(this.getCourse, this));
38    }
39
40    private getPeriod(period: string): Promise<Period> {
41      return getPeriod(this.db, period);
42    }
43
44    periods(): Promise<Period[]> {
45      return Promise.all(this._periods.map(this.getPeriod, this));
46    }
47  }
48
49  export async function getUniversity(
50    db: Database,
51    id: string
52  ): Promise<University> {
53    let row = await db.get(TableUniversity, id);
54    return new University(db, row);
55  }
56
```

```
57 export async function getAllUniversities(db: Database):
   ↪  Promise<University[]> {
58   let rows = await db.getAll(TableUniversity);
59   return rows.map(row => new University(db, row));
60 }
```

B.2.33 Pasta src/js/pages/account/components

Arquivo courses.tsx

```
1  import {
2    RemoteGetCoursesQuery,
3    RemoteGetUserDataFragment
4  } from "../../../graphql";
5  import { loadingOptions, fallback } from "./utilities";
6  import { ChipSet, Chip } from "@rmwc/chip";
7  import React from "react";
8  import { MultipleSelectEvents } from "../../base/components/form";
9
10 type CoursesListProps = {
11   loading: boolean;
12   data?: RemoteGetCoursesQuery;
13   account: RemoteGetUserDataFragment;
14   selected: string[];
15   events: MultipleSelectEvents;
16 };
17
18 export default function CoursesList(props: CoursesListProps) {
19   let { loading, data, account, selected, events } = props;
20   return (
21     <ChipSet choice>
22       {fallback(
23         loading || !data ? account?.courses ?? loadingOptions :
          ↪  data.courses,
24         {
25           label: "Selecione uma universidade",
26           value: ""
27         }
28       ).map(item => (
29         <Chip
```

```
            checkmark
            selected={selected.includes(item.value)}
            key={item.value}
            id={item.value}
            {...events}
            label={item.label}
          />
        ))}
      </ChipSet>
  );
}
```

Arquivo disciplines.tsx

```
import React, { Fragment } from "react";
import { RemoteGetDisciplinesByHabilitationsQuery } from
↪  "../../../graphql";
import { ChipSet, Chip } from "@rmwc/chip";
import { GridCell } from "@rmwc/grid";
import { Typography } from "@rmwc/typography";
import { MultipleSelectEvents } from "../../base/components/form";

type DisciplinesListProps = {
  data?: RemoteGetDisciplinesByHabilitationsQuery;
  selected: string[];
  events: MultipleSelectEvents;
};

export default function DisciplinesList(props: DisciplinesListProps) {
  let { data, selected, events } = props;
  return (
    <Fragment>
      {(data?.habilitations ?? []).map(habilitation => (
        <Fragment key={habilitation.id}>
          <GridCell span={12} className="form-row">
            <Typography use="headline6">
              {habilitation.course.name} - {habilitation.name}
            </Typography>
          </GridCell>
```

```
25            {habilitation.steps.map(step => (
26              <Fragment key={step.id}>
27                <GridCell span={12} className="form-row">
28                  {step.name}
29                </GridCell>
30                <GridCell span={12} className="form-row">
31                  <ChipSet choice>
32                    {step.disciplines.map(discipline => (
33                      <Chip
34                        checkmark
35                        selected={selected.includes(discipline.id)}
36                        key={discipline.id}
37                        {...events}
38                        label={discipline.code + " - " + discipline.name}
39                      />
40                    ))}
41                  </ChipSet>
42                </GridCell>
43              </Fragment>
44            ))}
45          </Fragment>
46        ))}
47      </Fragment>
48    );
49  }
```

Arquivo habilitations.tsx

```
1  import {
2    RemoteGetHabilitationsQuery,
3    RemoteGetUserDataFragment
4  } from "../../../graphql";
5  import { loadingOptions, fallback } from "./utilities";
6  import { ChipSet, Chip } from "@rmwc/chip";
7  import React from "react";
8  import { MultipleSelectEvents } from "../../base/components/form";
9
10 type HabiltiationsListProps = {
11   loading: boolean;
```

```
  data?: RemoteGetHabilitationsQuery;
  account: RemoteGetUserDataFragment;
  selected: string[];
  events: MultipleSelectEvents;
};

export default function HabilitationsList(props: HabiltiationsListProps)
↪ {
  let { loading, data, account, selected, events } = props;
  return (
    <ChipSet choice>
      {fallback(
        loading || !data
          ? account?.habilitations ?? loadingOptions
          : data.habilitations,
        {
          label: "Selecione um curso",
          value: ""
        }
      ).map(item => (
        <Chip
          checkmark
          selected={selected.includes(item.value)}
          key={item.value}
          {...events}
          label={item.label}
        />
      ))}
    </ChipSet>
  );
}
```

Arquivo index.tsx

```
import React, { useEffect, useMemo, Fragment } from "react";
import {
  useSubmit,
  useTextField,
  useSelect,
```

```
6     useMultipleSelect,
7     omit
8   } from "../../base/components/form";
9   import { Button } from "@rmwc/button";
10  import { Grid, GridCell } from "@rmwc/grid";
11  import { TextField } from "@rmwc/textfield";
12  import { Snackbar } from "@rmwc/snackbar";
13  import { useRedirectIfNotAuthenticated } from
    ↪  "../../base/components/auth";
14  import { Chip, ChipSet } from "@rmwc/chip";
15  import { useApolloClient, useQuery, useLazyQuery } from
    ↪  "@apollo/react-hooks";
16  import ApolloClient from "apollo-client";
17  import {
18    getAccountData,
19    getCourses,
20    getHabilitations,
21    getDisciplinesByHabilitations
22  } from "../queries/load-account.gql";
23  import { getUniversities } from "../../base/queries/universities.gql";
24  import {
25    RemoteGetUniversitiesQuery,
26    RemoteGetUniversitiesQueryVariables,
27    RemoteGetAccountDataQuery,
28    RemoteGetAccountDataQueryVariables,
29    RemoteGetCoursesQuery,
30    RemoteGetCoursesQueryVariables,
31    RemoteGetHabilitationsQuery,
32    RemoteGetHabilitationsQueryVariables,
33    RemoteGetDisciplinesByHabilitationsQuery,
34    RemoteGetDisciplinesByHabilitationsQueryVariables,
35    RemoteSaveAccountMutation,
36    RemoteSaveAccountMutationVariables,
37    RemoteUserDisciplineInput,
38    RemoteForeignKeyInput,
39    RemoteUserPasswordInput,
40    RemoteGetUserDataFragment
41  } from "../../../graphql";
42  import { saveAccount as mutation } from "../queries/save-account.gql";
```

```
43 import { Select } from "@rmwc/select";
44 import { QueryResult } from "@apollo/react-common";
45 import { Typography } from "@rmwc/typography";
46 import { FetchResult } from "apollo-link";
47 import { RemoteDisciplineDetails } from "../../discipline/components";
48 import { Option, loadingOptions } from "./utilities";
49 import CoursesList from "./courses";
50 import HabilitationsList from "./habilitations";
51 import DisciplinesList from "./disciplines";
52
53 type Fields = {
54   name: string;
55   oldPassword: string;
56   password: string;
57   confirmPassword: string;
58   universityID: string;
59   coursesIDs: string[];
60   habilitationsIDs: string[];
61   disciplinesIDs: string[];
62 };
63
64 type Data = {
65   universities: RemoteGetUniversitiesQuery["universities"];
66   courses: RemoteGetCoursesQuery["courses"];
67   habilitations: RemoteGetHabilitationsQuery["habilitations"];
68   disciplines: (RemoteUserDisciplineInput & Option)[];
69 };
70
71 async function submit<CacheShape>(
72   client: ApolloClient<CacheShape>,
73   data: Data,
74   fields: Fields
75 ): Promise<RemoteGetUserDataFragment> {
76   let result: FetchResult<RemoteSaveAccountMutation> | null;
77   let universityValue = data.universities.find(
78     university => university.value === fields.universityID
79   );
80   let university: RemoteForeignKeyInput | undefined;
81   if (universityValue !== undefined) {
```

```
82      university = { id: universityValue.value, name: universityValue.label
        ↪ };
83    }
84    let coursesMap: { [id: string]: RemoteGetCoursesQuery["courses"][0] } =
      ↪ {};
85    for (let course of data.courses) {
86      coursesMap[course.value] = course;
87    }
88    let courses: RemoteForeignKeyInput[] = [];
89    for (let courseID of fields.coursesIDs) {
90      let course = coursesMap[courseID];
91      if (!course) {
92        continue;
93      }
94      courses.push({
95        id: course.value,
96        name: course.label
97      });
98    }
99    let habilitationsMap: {
100     [id: string]: RemoteGetHabilitationsQuery["habilitations"][0];
101   } = {};
102   for (let habilitation of data.habilitations) {
103     habilitationsMap[habilitation.value] = habilitation;
104   }
105   let habilitations: RemoteForeignKeyInput[] = [];
106   for (let habilitationID of fields.habilitationsIDs) {
107     let habilitation = habilitationsMap[habilitationID];
108     if (!habilitation) {
109       continue;
110     }
111     habilitations.push({
112       id: habilitation.value,
113       name: habilitation.label
114     });
115   }
116   let disciplinesMap: { [id: string]: RemoteUserDisciplineInput & Option }
      ↪ = {};
117   for (let discipline of data.disciplines) {
```

```
118       disciplinesMap[discipline.id] = discipline;
119     }
120
121     let completedDisciplines: RemoteUserDisciplineInput[] = [];
122     for (let disciplineID of fields.disciplinesIDs) {
123       let discipline = disciplinesMap[disciplineID];
124       if (!discipline) {
125         continue;
126       }
127       completedDisciplines.push(
128         omit(discipline, ["__typename", "label", "value"])
129       );
130     }
131     let password: RemoteUserPasswordInput | undefined;
132     if (
133       fields.oldPassword.length > 0 ||
134       fields.password.length > 0 ||
135       fields.confirmPassword.length > 0
136     ) {
137       password = {
138         password: fields.password,
139         confirmPassword: fields.confirmPassword,
140         oldPassword: fields.oldPassword
141       };
142     }
143
144     try {
145       result = await client.mutate<
146         RemoteSaveAccountMutation,
147         RemoteSaveAccountMutationVariables
148       >({
149         mutation,
150         variables: {
151           user: {
152             name: fields.name,
153             university,
154             courses,
155             habilitations,
156             completedDisciplines,
```

```
157             password
158           }
159         }
160       });
161     } catch (e) {
162       result = null;
163     }
164     if (!result || !result.data) {
165       return Promise.reject("Erro inesperado durante a atualização do
          ↪ perfil");
166     }
167     return result.data.setUser;
168   }
169
170   type AccountFormProps = {
171     account: RemoteGetUserDataFragment;
172   };
173
174   function useCourse(
175     universityID: string
176   ): QueryResult<RemoteGetCoursesQuery, RemoteGetCoursesQueryVariables> {
177     let [fetchCourses, result] = useLazyQuery<
178       RemoteGetCoursesQuery,
179       RemoteGetCoursesQueryVariables
180     >(getCourses);
181     useEffect(
182       function() {
183         if (!universityID) {
184           return;
185         }
186         fetchCourses({
187           variables: {
188             universityID
189           }
190         });
191       },
192       [universityID, fetchCourses]
193     );
194     return result;
```

```
195 }
196
197 function getDisciplineValue(
198   discipline: RemoteGetUserDataFragment["completedDisciplines"][0]
199 ): Option {
200   return {
201     label: discipline.code + " - " + discipline.name,
202     value: discipline.id
203   };
204 }
205
206 function useHabilitation(
207   coursesIDs: string[]
208 ): QueryResult<
209   RemoteGetHabilitationsQuery,
210   RemoteGetHabilitationsQueryVariables
211 > {
212   let [fetchHabilitations, result] = useLazyQuery<
213     RemoteGetHabilitationsQuery,
214     RemoteGetHabilitationsQueryVariables
215   >(getHabilitations);
216
217   useEffect(
218     function() {
219       if (!coursesIDs) {
220         return;
221       }
222       fetchHabilitations({
223         variables: {
224           coursesIDs
225         }
226       });
227     },
228     [coursesIDs, fetchHabilitations]
229   );
230   return result;
231 }
232
233 function useDisciplines(
```

```
234   habilitationsIDs: string[]
235 ): QueryResult<
236   RemoteGetDisciplinesByHabilitationsQuery,
237   RemoteGetDisciplinesByHabilitationsQueryVariables
238 > {
239   let [fetchDisciplines, result] = useLazyQuery<
240     RemoteGetDisciplinesByHabilitationsQuery,
241     RemoteGetDisciplinesByHabilitationsQueryVariables
242   >(getDisciplinesByHabilitations);
243   useEffect(
244     function() {
245       if (!habilitationsIDs) {
246         return;
247       }
248       fetchDisciplines({
249         variables: {
250           habilitationsIDs
251         }
252       });
253     },
254     [habilitationsIDs, fetchDisciplines]
255   );
256   return result;
257 }
258
259 function AccountForm({ account }: AccountFormProps) {
260   const [name, nameField] = useTextField(account?.name);
261   const [oldPassword, oldPasswordField] = useTextField();
262   const [password, passwordField] = useTextField();
263   const [confirmPassword, confirmPasswordField] = useTextField();
264   const [universityID, universityField] =
      ↪ useSelect(account?.university?.value);
265   let { data: courses, loading: coursesLoading } =
      ↪ useCourse(universityID);
266   const { selected: coursesIDs, events: eventCourses } =
      ↪ useMultipleSelect(
267     account?.courses,
268     courses?.courses
269   );
```

```
  let { data: habilitations, loading: habilitationsLoading } =
  ↪  useHabilitation(
    coursesIDs
  );
  const {
    selected: habilitationsIDs,
    events: eventHabilitations
  } = useMultipleSelect(account?.habilitations,
  ↪  habilitations?.habilitations);

  let { data: disciplines, loading: disciplinesLoading } =
  ↪  useDisciplines(
    habilitationsIDs
  );
  let disciplinesList = useMemo(
    function() {
      let values: (Option & RemoteUserDisciplineInput)[] = [];
      if (!disciplines) {
        return values;
      }
      for (let habilitation of disciplines.habilitations) {
        for (let step of habilitation.steps) {
          for (let discipline of step.disciplines) {
            values.push({
              ...discipline,
              value: discipline.id,
              label: discipline.name
            });
          }
        }
      }
      return values;
    },
    [disciplines]
  );
  const {
    selected: disciplinesIDs,
    events: eventDisciplines
  } = useMultipleSelect(
```

```
306       (account?.completedDisciplines ?? []).map(getDisciplineValue),
307       disciplinesList
308     );
309     const client = useApolloClient();
310     let { data: universities, loading: universitiesLoading } = useQuery<
311       RemoteGetUniversitiesQuery,
312       RemoteGetUniversitiesQueryVariables
313     >(getUniversities);
314
315     const { onSubmit, submitted, error, clearError } = useSubmit(() =>
316       submit(
317         client,
318         {
319           courses: courses?.courses ?? [],
320           disciplines: disciplinesList,
321           habilitations: habilitations?.habilitations ?? [],
322           universities: universities?.universities ?? []
323         },
324         {
325           name,
326           coursesIDs,
327           habilitationsIDs,
328           disciplinesIDs,
329           universityID,
330           password,
331           confirmPassword,
332           oldPassword
333         }
334       )
335     );
336     if (submitted) {
337       return (
338         <Grid>
339           <GridCell span={12} className="form-row">
340             Conta editada com sucesso!
341           </GridCell>
342         </Grid>
343       );
344     }
```

```
345
346   return (
347     <form onSubmit={onSubmit}>
348       <Grid>
349         <GridCell span={12} className="form-row">
350           <TextField
351             name="name"
352             type="text"
353             label="Nome"
354             icon="person"
355             required
356             {...nameField}
357           />
358         </GridCell>
359         <GridCell span={12} className="form-row">
360           <TextField
361             name="old-password"
362             type="password"
363             label="Senha Antiga:"
364             icon="security"
365             {...oldPasswordField}
366           />
367         </GridCell>
368         <GridCell span={12} className="form-row">
369           <TextField
370             name="password"
371             type="password"
372             label="Senha"
373             icon="security"
374             {...passwordField}
375           />
376         </GridCell>
377         <GridCell span={12} className="form-row">
378           <TextField
379             name="password"
380             type="password"
381             label="Confirmar Senha"
382             icon="security"
383             {...confirmPasswordField}
```

```
384           />
385         </GridCell>
386         <GridCell span={12} className="form-row">
387           <Select
388             name="university"
389             label="Universidade"
390             icon="school"
391             required
392             enhanced
393             {...universityField}
394             options={
395               universitiesLoading || !universities
396                 ? [account?.university ?? loadingOptions[0]]
397                 : universities.universities
398             }
399           />
400         </GridCell>
401         <GridCell span={12} className="form-row">
402           <Typography use="headline5">Cursos:</Typography>
403         </GridCell>
404         <GridCell span={12} className="form-row">
405           <CoursesList
406             account={account}
407             data={courses}
408             selected={coursesIDs}
409             loading={coursesLoading}
410             events={eventCourses}
411           />
412         </GridCell>
413         <GridCell span={12} className="form-row">
414           <Typography use="headline5">Habilitações:</Typography>
415         </GridCell>
416         <GridCell span={12} className="form-row">
417           <HabilitationsList
418             account={account}
419             data={habilitations}
420             selected={habilitationsIDs}
421             loading={habilitationsLoading}
422             events={eventHabilitations}
```

```
423             />
424           </GridCell>
425           <GridCell span={12} className="form-row">
426             <Typography use="headline5">Disciplinas:</Typography>
427           </GridCell>
428           <DisciplinesList
429             data={disciplines}
430             selected={disciplinesIDs}
431             events={eventDisciplines}
432           />
433           <GridCell span={12}>
434             <Button type="submit" raised>
435               Salvar
436             </Button>
437           </GridCell>
438         </Grid>
439         {error ? <Snackbar open onClose={clearError} message={error} /> :
            ↪ null}
440       </form>
441     );
442   }
443   export default function Profile() {
444     useRedirectIfNotAuthenticated();
445     let { data, error, loading } = useQuery<
446       RemoteGetAccountDataQuery,
447       RemoteGetAccountDataQueryVariables
448     >(getAccountData);
449     if (error) {
450       return <p>Houve um erro durante o carregamento do perfil..</p>;
451     }
452     if (loading) {
453       return <p>Carregando...</p>;
454     }
455     if (!data || !data.loggedUser) {
456       return <p>Usuário não encontrado.</p>;
457     }
458     return <AccountForm account={data.loggedUser} />;
459   }
```

Arquivo utilities.ts

```
import { ChipOnInteractionEventT } from "@rmwc/chip";
import { MouseEvent } from "react";

export const loadingOptions = [
  {
    label: "Carregando...",
    value: ""
  }
];

export type Option = {
  value: string;
  label: string;
};

export function fallback(options: Option[], fallback: Option) {
  return options.length === 0 ? [fallback] : options;
}
```

B.2.34 Pasta src/js/pages/account/queries

Arquivo load-account.gql

```
fragment getUserData on User {
  id
  name
  email
  oauth2_provider
  university {
    value: id
    label: name
  }
  courses {
    value: id
    label: name
  }
  habilitations {
    value: id
```

```
    label: name
  }
  completedDisciplines {
    id
    genericID
    code
    name
  }
}

query getAccountData {
  loggedUser {
    ...getUserData
  }
}

query getCourses($universityID: ID!) {
  courses: coursesByUniversity(universityID: $universityID) {
    value: id
    label: name
  }
}

query getHabilitations($coursesIDs: [ID!]!) {
  habilitations: habilitationsByCourses(coursesIDs: $coursesIDs) {
    value: id
    label: name
  }
}

query getDisciplinesByHabilitations($habilitationsIDs: [ID!]!) {
  habilitations: habilitationsByIDs(ids: $habilitationsIDs) {
    id
    name
    course {
      id
      name
    }
    steps {
```

```
      id
      name
      disciplines {
        id
        genericID
        code
        name
      }
    }
  }
}
```

Arquivo save-account.gql

```
#import "./load-account.gql"

mutation saveAccount($user: UserInput!) {
  setUser(user: $user) {
    ...getUserData
  }
}
```

B.2.35 Pasta src/js/pages/base/components

Arquivo auth.ts

```
import { useDispatch, useSelector } from "react-redux";
import { getQueryString } from "../store/querystring";
import { useEffect, useCallback } from "react";
import { getAuthenticatedUser } from "../store/auth";
import { push, RouterRootState, getLocation } from
↪  "connected-react-router";
import { Location } from "history";

const isAbsoluteRegex = /^(?:[a-z]+:)?/;

export function useRedirectIfAuthenticated(
  paramName: string = "redir",
  defaultPath: string = "/"
) {
```

```
14    const dispatch = useDispatch();
15    const querystring = useSelector(getQueryString);
16    const user = useSelector(getAuthenticatedUser);
17    let param = querystring[paramName];
18    useEffect(() => {
19      if (!user) {
20        return;
21      }
22      let redirectTo = defaultPath;
23      if (param && !Array.isArray(param) && isAbsoluteRegex.test(param)) {
24        redirectTo = param;
25      }
26      dispatch(push(redirectTo));
27    }, [dispatch, user, param]);
28  }
29
30  function redirectCallback(basePath: string, paramName: string = "redir")
    ↪ {
31    const dispatch = useDispatch();
32    const actual = useSelector<RouterRootState, Location>(getLocation);
33    const path = encodeURIComponent(actual.pathname + actual.search);
34    const callback = useCallback(
35      () => dispatch(push(`${basePath}?${paramName}=${path}`)),
36      [dispatch, basePath, paramName, path]
37    );
38    return callback;
39  }
40
41  export function useLoginCallback() {
42    return redirectCallback("/entrar");
43  }
44
45  export function useSignupCallback() {
46    return redirectCallback("/cadastro");
47  }
48
49  export function useRedirectIfNotAuthenticated() {
50    let callback = useLoginCallback();
51    const user = useSelector(getAuthenticatedUser);
```

```
52    useEffect(() => {
53      if (user) {
54        return;
55      }
56      callback();
57    }, [callback, user]);
58  }
```

Arquivo form.tsx

```
1   import {
2     ChangeEvent,
3     FormEvent,
4     useState,
5     useCallback,
6     useEffect,
7     MouseEvent,
8     useMemo
9   } from "react";
10  import { useSelector } from "react-redux";
11  import { getCSRFToken } from "../store/csrf/selectors";
12  import { ChipOnInteractionEventT } from "@rmwc/chip";
13
14  export interface SubmitResponse {
15    submitted: boolean;
16    error?: string;
17  }
18
19  interface HTMLElementValue {
20    value: string;
21  }
22
23  interface Valued {
24    value: string;
25  }
26
27  function getValue(el: Valued): string {
28    return el.value;
29  }
30
```

```
function useValue<T extends HTMLElementValue>(
  initialValue?: string,
  callback?: (value: string) => void
): [string, { value: string; onChange: (evt: ChangeEvent<T>) => void }] {
  let [value, setValue] = useState(initialValue ?? "");
  let onChange = useCallback(
    (evt: ChangeEvent<T>) => {
      let newValue = getValue(evt.target);
      setValue(newValue);
      if (callback) {
        callback(newValue);
      }
    },
    [callback]
  );
  return [value, { value, onChange }];
}

export function useTextField(
  initialValue?: string,
  callback?: (value: string) => void
): [
  string,
  { value: string; onChange: (evt: ChangeEvent<HTMLInputElement>) => void
  ↪ }
] {
  return useValue(initialValue, callback);
}

export function useSelect(
  initialValue?: string,
  callback?: (value: string) => void
): [
  string,
  { value: string; onChange: (evt: ChangeEvent<HTMLSelectElement>) =>
  ↪ void }
] {
  return useValue(initialValue, callback);
}
```

```
68
69  function prevent(evt: MouseEvent) {
70    evt.preventDefault();
71  }
72
73  export function useChipSelect(
74    initialValue: Valued
75  ): {
76    selected: string;
77    onClick: (evt: MouseEvent) => void;
78    onInteraction: (evt: ChipOnInteractionEventT) => void;
79  } {
80    let [selected, setSelected] = useState(getValue(initialValue));
81    return {
82      selected,
83      onClick: prevent,
84      onInteraction(evt: ChipOnInteractionEventT) {
85        let { chipId: value } = evt.detail;
86        setSelected(value);
87      }
88    };
89  }
90
91  export type MultipleSelectEvents = {
92    onClick: (evt: MouseEvent) => void;
93    onInteraction: (evt: ChipOnInteractionEventT) => void;
94  };
95
96  export function useMultipleSelect(
97    initial?: Valued[],
98    values?: Valued[]
99  ): {
100   selected: string[];
101   events: MultipleSelectEvents;
102 } {
103   let [selected, setSelected] = useState((initial ?? []).map(getValue));
104   useEffect(
105     function() {
106       let ids = values ? values.map(getValue) : [];
```

```
      if (ids.length === 0) {
        // Probably the data is still loading..so don't mess with it
        return;
      }
      let newSelected = selected.filter(function(id) {
        return ids.includes(id);
      });
      if (newSelected.length < selected.length) {
        setSelected(newSelected);
      }
    },
    [values]
  );
  return {
    selected,
    events: useMemo(
      function() {
        return {
          onClick: prevent,
          onInteraction(evt: ChipOnInteractionEventT) {
            let { chipId: value } = evt.detail;
            if (selected.includes(value)) {
              setSelected(selected.filter(item => item !== value));
            } else {
              setSelected(selected.concat([value]));
            }
          }
        };
      },
      [selected]
    )
  };
}

export function useCheckboxField(
  initialValue?: boolean,
  callback?: (evt: ChangeEvent<HTMLInputElement>) => void
): [
  boolean,
```

```
  { checked: boolean; onChange: (evt: ChangeEvent<HTMLInputElement>) =>
  ↪ void }
] {
  let [checked, setChecked] = useState(initialValue === true);
  let onChange = useCallback(
    (evt: ChangeEvent<HTMLInputElement>) => {
      setChecked(evt.target.checked);
      if (callback) {
        callback(evt);
      }
    },
    [callback]
  );
  return [checked, { checked, onChange }];
}

interface SubmitState<T> {
  submitted: T | null;
  error: string | null;
}

export function useSubmit<T extends {}>(
  handleSubmit: (
    evt: FormEvent<HTMLFormElement>,
    csrfToken: string | null
  ) => Promise<T>,
  error: string | null = null
) {
  let csrfToken = useSelector(getCSRFToken);
  let [formState, setFormState] = useState<SubmitState<T>>({
    submitted: null,
    error
  });

  function onSubmit(evt: FormEvent<HTMLFormElement>) {
    evt.preventDefault();
    let form = evt.target as HTMLFormElement;
    if (form.reportValidity()) {
      handleSubmit(evt, csrfToken).then(
```

```
184           submitted => setFormState({ submitted, error: null }),
185           showError
186         );
187       }
188     }
189     let setError = useCallback((error: string | null) => {
190       setFormState({
191         submitted: null,
192         error
193       });
194     }, []);
195
196     let showError = useCallback((error: string) => setError(error), []);
197     let clearError = useCallback(() => setError(null), []);
198     return {
199       ...formState,
200       onSubmit,
201       showError,
202       clearError
203     };
204   }
205
206   export function omit(obj: any, fields: string[]): any {
207     return Object.entries(obj)
208       .filter(([key]) => !fields.includes(key))
209       .reduce((obj, [key, val]) => {
210         if (Array.isArray(val)) {
211           val = val.map(item => omit(item, fields));
212         } else if (val !== null && typeof val === "object") {
213           val = omit(val, fields);
214         }
215         obj[key] = val;
216         return obj;
217       }, {});
218   }
```

Arquivo infinite-scroll.tsx

```
1   import React, {
2     useEffect,
```

```
  useCallback,
  useRef,
  ReactNode,
  useState,
  MutableRefObject,
  Fragment,
  useReducer,
  useLayoutEffect
} from "react";
import { Index } from "lunr";

export type InfiniteScrollRenderOptions<T> = {
  items: T[];
  loading: boolean;
  start: number;
  end: number;
  itemCount: number;
};

type InfiniteScrollProps<T, U> = {
  itemCount: number;
  items: T[];
  loading: boolean;
  batch?: number;
  options?: IntersectionObserverInit;
  loadMore: () => void;
  render: (options: InfiniteScrollRenderOptions<T> & U) => ReactNode;
  renderOptions: U;
};

type ObserverOptions = {
  beforeRef: MutableRefObject<HTMLDivElement | null>;
  afterRef: MutableRefObject<HTMLDivElement | null>;
  callback: IntersectionObserverCallback;
  options?: IntersectionObserverInit;
};

function useObserver(observerOptions: ObserverOptions) {
  const { options, afterRef, beforeRef, callback } = observerOptions;
```

```
  useEffect(
    function() {
      if (!beforeRef.current || !afterRef.current) {
        return;
      }
      let observer = new IntersectionObserver(callback, options);
      observer.observe(beforeRef.current);
      observer.observe(afterRef.current);
      return function() {
        observer.disconnect();
      };
    },
    [beforeRef, afterRef, callback, options]
  );
}

type Dimension = {
  width: number;
  height: number;
};
type SliceState = {
  start: number;
  end: number | null;
  totalHeight: number;
  beforeHeight: number;
  contentHeight: number;
  afterHeight: number;
};

type SliceAction =
  | {
      type: "up" | "down" | "both";
      length: number;
      batch: number;
    }
  | {
      type: "resize";
      totalHeight: number;
    };
```

```
function SliceReducer(state: SliceState, action: SliceAction): SliceState {
  if (action.type === "resize") {
    let { totalHeight } = action;
    if (totalHeight > state.totalHeight) {
      return { ...state, totalHeight };
    }
    return state;
  }
  let { length, batch } = action;
  let end = state.end === null ? action.length : state.end;
  switch (action.type) {
    case "up":
      if (state.start - batch < 0) {
        return state;
      }
      return {
        ...state,
        start: state.start - batch,
        end: end - batch
      };
    case "down":
      if (end + batch >= length) {
        return state;
      }
      return {
        ...state,
        start: state.start + batch,
        end: end + batch
      };
    case "both":
      return {
        ...state,
        start: Math.max(state.start - batch, 0),
        end: Math.min(end + batch, length)
      };
    default:
      throw new Error("Unknown action");
```

```
119   }
120 }
121
122 function InfiniteScroll<T, U>(props: InfiniteScrollProps<T, U>) {
123   const [slice, dispatch] = useReducer(SliceReducer, {
124     start: 0,
125     end: null,
126     totalHeight: 2,
127     beforeHeight: 1,
128     afterHeight: 1,
129     contentHeight: 0
130   });
131   const containerRef = useRef<HTMLDivElement | null>(null);
132   const contentRef = useRef<HTMLDivElement | null>(null);
133   const beforeRef = useRef<HTMLDivElement | null>(null);
134   const afterRef = useRef<HTMLDivElement | null>(null);
135   const { items, loading, itemCount, options, renderOptions, loadMore } =
      ↪  props;
136   let batch = !props.batch ? 1 : props.batch;
137   const callback = useCallback(
138     function(entries: IntersectionObserverEntry[]) {
139       if (itemCount === 0) {
140         return;
141       }
142       let after = false;
143       let before = false;
144       for (let entry of entries) {
145         if (!entry.isIntersecting) {
146           continue;
147         }
148         // TODO: find a way to virtualize list by setting beforeHeight
149         // and afterHeight correctly and using it as placeholders for
150         // the elements hidden in the list
151         after = after || entry.target === afterRef.current;
152         before = before || entry.target === beforeRef.current;
153       }
154       let shouldLoadMore = itemCount > items.length && !loading;
155       if (after && shouldLoadMore) {
156         loadMore();
```

```
      }
      let { length } = items;
      let end = slice.end ?? batch;
      if (after && before) {
        if (shouldLoadMore) {
          return;
        }
        dispatch({ type: "both", length, batch });
      } else if (after) {
        if (end === itemCount) {
          // We're already at the last element, so ignore it right now
          return;
        }
        dispatch({ type: "down", length, batch });
      } else if (before) {
        if (slice.start === 0) {
          // We're already at the first element, so ignore it right now
          return;
        }
        dispatch({ type: "up", length, batch });
      }
    },
    [itemCount, items.length, loading, afterRef, slice, dispatch,
    ↪ loadMore]
  );
  useObserver({
    afterRef,
    beforeRef,
    callback,
    options
  });
  useLayoutEffect(function() {});
  return (
    <div ref={containerRef}>
      <div style={{ height: slice.beforeHeight + "px" }}
      ↪ ref={beforeRef}></div>
      <div ref={contentRef}>
        {props.render({
          ...renderOptions,
```

```
            items,
            start: slice.start,
            end: slice.end ?? items.length,
            loading,
            itemCount
          })}
        </div>
        <div style={{ height: slice.afterHeight + "px" }}
        ↪  ref={afterRef}></div>
      </div>
  );
}

export default InfiniteScroll;
```

Arquivo static.tsx

```
import React from "react";
import { Grid, GridCell } from "@rmwc/grid";

type RelatedItem = {
  url: string;
  title: string;
};

type Single = {
  title: string;
  content: string;
  date: Date;
  author: {
    username: string;
    name: string;
    url: string;
  };
  related: RelatedItem[];
};

type ListItem = {
  title: string;
```

```
23    url: string;
24    summary: string;
25    truncated: boolean;
26    author: {
27      username: string;
28      name: string;
29      url: string;
30    };
31    date: Date;
32  };
33
34  type Page = {
35    actual: number;
36    first: string;
37    last: string;
38    next: string | null;
39    prev: string | null;
40    total_elements: number;
41    total_pages: number;
42  };
43
44  type List = {
45    items: ListItem[];
46  };
47
48  type SingleComponentProps = {
49    single: Single;
50  };
51
52  function formatDate(date?: Date): string {
53    if (!date) {
54      return "Nunca";
55    }
56    return new Date(date).toLocaleDateString("pt-br", {
57      day: "numeric",
58      weekday: "long",
59      year: "numeric",
60      month: "numeric",
61      hour: "numeric",
```

```
    minute: "numeric"
  });
}

export function SingleComponent(props: SingleComponentProps) {
  return (
    <Grid>
      <GridCell span={12}>{props.single.content}</GridCell>
      <GridCell span={12}>Postado em
      ↪ {formatDate(props.single.date)}</GridCell>
      <GridCell span={12}>Postado por
      ↪ {props.single.author.name}</GridCell>
    </Grid>
  );
}
```

Arquivo table.tsx

```
import React from "react";
import { RemoteTableFragment, RemoteTableColumn } from
↪ "../../../graphql";
import {
  DataTable,
  DataTableContent,
  DataTableHead,
  DataTableRow,
  DataTableHeadCell,
  DataTableBody,
  DataTableCell
} from "@rmwc/data-table";
import { GridCell, GridInner } from "@rmwc/grid";
import { Typography } from "@rmwc/typography";

type RowProps = {
  row: { [column: number]: string };
  columns: RemoteTableColumn[];
};

function Row({ columns, row }: RowProps) {
```

```
  return (
    <DataTableRow>
      {columns.map((column, index) =>
        row[index] ? (
          <DataTableCell key={column.id}>{row[index]}</DataTableCell>
        ) : null
      )}
    </DataTableRow>
  );
}

export type TableProps = {
  table: RemoteTableFragment;
};

export function Table({ table }: TableProps) {
  let rows: { [row: number]: { [column: number]: string } } = {};
  let rowsKeys: number[] = [];
  for (let { row, column, value } of table.rows) {
    if (!rows[row]) {
      rows[row] = {};
      rowsKeys.push(row);
    }
    rows[row][column] = value;
  }
  return (
    <GridInner>
      <GridCell span={12}>
        <Typography use="headline5">{table.title}</Typography>
      </GridCell>
      {table.description ? (
        <GridCell span={12}>
          <Typography use="body1">{table.description}</Typography>
        </GridCell>
      ) : null}
      <GridCell span={12}>
        <DataTable>
          <DataTableContent>
            <DataTableHead>
```

```
60                <DataTableRow>
61                  {table.columns.map(column => (
62                    <DataTableHeadCell key={column.id}>
63                      {column.name}
64                    </DataTableHeadCell>
65                  ))}
66                </DataTableRow>
67              </DataTableHead>
68              <DataTableBody>
69                {rowsKeys.map(rowKey => (
70                  <Row key={rowKey} row={rows[rowKey]}
                  ↪  columns={table.columns} />
71                ))}
72              </DataTableBody>
73            </DataTableContent>
74          </DataTable>
75        </GridCell>
76      </GridInner>
77    );
78  }
```

Arquivo title.tsx

```
1  import React, { Component, ReactNode, useEffect } from "react";
2  import { useContext } from "react";
3
4  interface Props {
5    title: string;
6    children: ReactNode;
7  }
8
9  const TitleContext = React.createContext([] as string[]);
10
11 const separator = " - ";
12
13 function setTitle(...title: string[]) {
14   document.title = title.join(separator);
15 }
16 export default function Title(props: Props) {
```

```
17    let title = useContext(TitleContext);
18    useEffect(() => {
19      setTitle(props.title, ...title);
20      return () => {
21        setTitle(...title);
22      };
23    }, [props.title, title]);
24    let newTitle: string[] = [props.title, ...title];
25    return (
26      <TitleContext.Provider value={newTitle}>
27        {props.children}
28      </TitleContext.Provider>
29    );
30  }
```

B.2.36 Pasta src/js/pages/base/queries

Arquivo periods.gql

```
1  query getPeriods($universityID: ID!) {
2    periods: periodsByUniversity(universityID: $universityID) {
3      value: id
4      label: name
5    }
6  }
```

Arquivo table.gql

```
1  fragment table on Table {
2    id
3    title
4    description
5    columns {
6      id
7      name
8    }
9    rows {
10     row
11     column
12     value
```

```
  }
}
```

Arquivo universities.gql

```
query getUniversities {
  universities {
    value: id
    label: name
  }
}
```

Arquivo user.gql

```
query getUser {
  loggedUser {
    id
    name
    email
    provider: oauth2_provider
  }
}
```

B.2.37 Pasta src/js/pages/base/store

Arquivo index.ts

```
import { IModule } from "redux-dynamic-modules";
import getUIModule, { FullState as UIState } from "./ui";
import getAuthModule, { FullState as AuthState } from "./auth";
import { FullState as CSRFState } from "./csrf";

export default function getReduxModule(): IModule<
  UIState | AuthState | CSRFState
>[] {
  return [getUIModule(), ...getAuthModule()];
}
```

B.2.38 Pasta src/js/pages/create-plan/components

Arquivo index.tsx

```
import React, { ChangeEvent, useEffect } from "react";
import { useSubmit, useTextField, useSelect } from
↪ "../../base/components/form";
import { Button } from "@rmwc/button";
import { Grid, GridCell } from "@rmwc/grid";
import { TextField } from "@rmwc/textfield";
import { Select } from "@rmwc/select";
import { Snackbar } from "@rmwc/snackbar";
import { useRedirectIfNotAuthenticated } from
↪ "../../base/components/auth";
import { useApolloClient, useQuery, useLazyQuery } from
↪ "@apollo/react-hooks";
import ApolloClient from "apollo-client";
import { createPlan } from "../queries/create-plan.gql";
import {
  RemoteCreatePlanMutation,
  RemoteCreatePlanMutationVariables,
  RemoteGetPeriodsQuery,
  RemoteGetPeriodsQueryVariables,
  RemoteGetUniversitiesQuery,
  RemoteGetUniversitiesQueryVariables
} from "../../../graphql";
import { getUniversities } from "../../base/queries/universities.gql";
import { getPeriods } from "../../base/queries/periods.gql";
import { useDispatch } from "react-redux";
import { redirect } from "../../base/store/querystring";

type Fields = {
  name: string;
  periodID: string;
  universityID: string;
};

async function submit<CacheShape>(
  client: ApolloClient<CacheShape>,
  fields: Fields
): Promise<RemoteCreatePlanMutation> {
  if (fields.universityID === "-" || fields.periodID === "-") {
    return Promise.reject("Selecione uma universidade e um período");
```

```
37    }
38    let result = await client.mutate<
39      RemoteCreatePlanMutation,
40      RemoteCreatePlanMutationVariables
41    >({
42      mutation: createPlan,
43      variables: {
44        plan: {
45          ...fields,
46          disciplines: [],
47          offers: [],
48          possibilities: [
49            {
50              name: "Possibilidade 1",
51              selectionDisciplines: [],
52              selectionTeams: []
53            }
54          ],
55          public: false,
56          selectedPossibility: 0
57        }
58      }
59    });
60    if (!result.data) {
61      return Promise.reject("Erro inesperado durante o cadastro da
      ↪  universidade");
62    }
63    return result.data;
64  }
65
66  export default function CreatePlan() {
67    useRedirectIfNotAuthenticated();
68    const [name, nameField] = useTextField();
69    let {
70      data: universitiesData,
71      loading: universitiesLoading,
72      error: universitiesError
73    } = useQuery<RemoteGetUniversitiesQuery,
      ↪  RemoteGetUniversitiesQueryVariables>(
```

```
74      getUniversities
75    );
76    const [
77      getPeriodsCallback,
78      { loading: periodsLoading, data: periodsData, error: periodsError }
79    ] = useLazyQuery<RemoteGetPeriodsQuery,
      ↪  RemoteGetPeriodsQueryVariables>(
80      getPeriods
81    );
82    const [universityID, universityIDField] = useSelect("-", function(
83      newUniversityID
84    ) {
85      getPeriodsCallback({
86        variables: {
87          universityID: newUniversityID
88        }
89      });
90    });
91    const [periodID, periodIDField] = useSelect("-");
92    let universities = [
93      {
94        value: "-",
95        label: universitiesError
96          ? "Erro ao carregar dados "
97          : universitiesLoading
98          ? "Carregando..."
99          : "Selecione uma universidade"
100     }
101   ];
102   let periods = [
103     {
104       value: "-",
105       label: periodsError
106         ? "Erro ao carregar dados"
107         : periodsLoading
108         ? "Carregando..."
109         : universityID === "-"
110         ? "Selecione uma universidade primeiro"
111         : "Selecione um período"
```

```
112       }
113     ];
114     if (universitiesData) {
115       universities.push(...universitiesData.universities);
116     }
117     if (periodsData) {
118       periods.push(...periodsData.periods);
119     }
120     const client = useApolloClient();
121     const { onSubmit, submitted, error, clearError } = useSubmit(() =>
122       submit(client, {
123         name,
124         universityID,
125         periodID
126       })
127     );
128     let dispatch = useDispatch();
129     useEffect(
130       function() {
131         if (!submitted || !submitted.setPlan || !submitted.setPlan.version)
          ↪ {
132           return;
133         }
134         dispatch(
135           redirect({
136             url: "/planos/editar",
137             querystring: {
138               id: submitted.setPlan.id,
139               version: submitted.setPlan.version.id
140             }
141           })
142         );
143       },
144       [submitted]
145     );
146     if (submitted) {
147       if (!submitted.setPlan || !submitted.setPlan.version) {
148         return <p>Houve um erro ao criar o plano.</p>;
149       }
```

```
150     return <p>Plano criado!</p>;
151   }
152   return (
153     <form onSubmit={onSubmit}>
154       <Grid>
155         <GridCell span={12} className="form-row">
156           <TextField
157             name="name"
158             type="text"
159             label="Nome"
160             icon="school"
161             required
162             {...nameField}
163           />
164         </GridCell>
165         <GridCell span={12} className="form-row">
166           <Select
167             name="university"
168             label="Universidade"
169             icon="school"
170             required
171             enhanced
172             options={universities}
173             {...universityIDField}
174           />
175         </GridCell>
176         <GridCell span={12} className="form-row">
177           <Select
178             name="period"
179             label="Período:"
180             icon="event"
181             required
182             enhanced
183             options={periods}
184             {...periodIDField}
185           />
186         </GridCell>
187         <GridCell span={12} className="form-row">
188           <Button type="submit" raised>
```

```
189                 Criar
190               </Button>
191             </GridCell>
192           </Grid>
193           {error ? <Snackbar open onClose={clearError} message={error} /> :
              ↪  null}
194         </form>
195       );
196     }
```

B.2.39 Pasta src/js/pages/create-plan/queries

Arquivo create-plan.gql

```
1  mutation createPlan($plan: PlanInput!) {
2    setPlan(plan: $plan) {
3      id
4      version: versionByIndex(index: -1) {
5        id
6      }
7    }
8  }
```

B.2.40 Pasta src/js/pages/create-university/components

Arquivo index.tsx

```
1  import React from "react";
2  import { useSubmit, useTextField } from "../../base/components/form";
3  import { Button } from "@rmwc/button";
4  import { Grid, GridCell } from "@rmwc/grid";
5  import { TextField } from "@rmwc/textfield";
6  import { Snackbar } from "@rmwc/snackbar";
7  import { useRedirectIfNotAuthenticated } from
   ↪  "../../base/components/auth";
8  import { Typography } from "@rmwc/typography";
9  import { useApolloClient } from "@apollo/react-hooks";
10 import ApolloClient from "apollo-client";
11 import mutation from "../queries/create-university.gql";
12 import {
```

```
13    RemoteCreateUniversityMutation,
14    RemoteCreateUniversityMutationVariables
15  } from "../../../graphql";
16  import { FetchResult } from "apollo-link";
17
18  type FieldsData = {
19    baseUrl: string;
20    deleteUrl: string;
21    aboutUrl: string;
22  };
23
24  type Fields = {
25    acronym: string;
26    name: string;
27    offers: FieldsData;
28    habilitations: FieldsData;
29  };
30
31  async function submit<CacheShape>(
32    client: ApolloClient<CacheShape>,
33    fields: Fields
34  ): Promise<RemoteCreateUniversityMutation> {
35    let result: FetchResult<RemoteCreateUniversityMutation> | null;
36    let regex = /\{filename\}/;
37    if (!regex.test(fields.offers.baseUrl)) {
38      return Promise.reject(
39        "A URL para download dos dados de ofertas precisa possuir
          ↪ {filename}"
40      );
41    }
42    if (!regex.test(fields.habilitations.baseUrl)) {
43      return Promise.reject(
44        "A URL para download dos dados de cursos e habilitações precisa
          ↪  possuir {filename}"
45      );
46    }
47    try {
48      result = await client.mutate<
49        RemoteCreateUniversityMutation,
```

```
50        RemoteCreateUniversityMutationVariables
51      >({
52        mutation,
53        variables: {
54          university: {
55            acronym: fields.acronym,
56            name: fields.name,
57            offers: {
58              ...fields.offers,
59              generateSecret: true
60            },
61            habilitations: {
62              ...fields.habilitations,
63              generateSecret: true
64            },
65            public: false
66          }
67        }
68      });
69    } catch (e) {
70      result = null;
71    }
72    if (!result || !result.data) {
73      return Promise.reject("Erro inesperado durante o cadastro da
      ↪  universidade");
74    }
75    return result.data;
76  }
77
78  export default function CreateUniversity() {
79    const [acronym, acronymField] = useTextField();
80    const [name, nameField] = useTextField();
81    const [baseURLOffers, baseURLOffersField] = useTextField();
82    const [deleteURLOffers, deleteURLOffersField] = useTextField();
83    const [aboutURLOffers, aboutURLOffersField] = useTextField();
84    const [baseURLHabilitations, baseURLHabilitationsField] =
      ↪  useTextField();
85    const [deleteURLHabilitations, deleteURLHabilitationsField] =
      ↪  useTextField();
```

```
86    const [aboutURLHabilitations, aboutURLHabilitationsField] =
      ↪ useTextField();
87    useRedirectIfNotAuthenticated();
88    const client = useApolloClient();
89    const { onSubmit, submitted, error, clearError } = useSubmit(() =>
90      submit(client, {
91        acronym,
92        name,
93        offers: {
94          baseUrl: baseURLOffers,
95          aboutUrl: aboutURLOffers,
96          deleteUrl: deleteURLOffers
97        },
98        habilitations: {
99          baseUrl: baseURLHabilitations,
100         aboutUrl: aboutURLHabilitations,
101         deleteUrl: deleteURLHabilitations
102       }
103     })
104   );
105   if (submitted) {
106     return (
107       <Grid>
108         <GridCell span={12}>Universidade cadastrada!</GridCell>
109         <GridCell span={6}>O segredo para as ofertas de turmas
          ↪ é:</GridCell>
110         <GridCell span={6}>
111           <TextField value={submitted.setUniversity.offers.secret}
              ↪ readOnly />
112         </GridCell>
113         <GridCell span={6}>O segredo para os dados de habilitações
          ↪ é:</GridCell>
114         <GridCell span={6}>
115           <TextField
116             value={submitted.setUniversity.habilitations.secret}
117             readOnly
118           />
119         </GridCell>
120       </Grid>
```

```
121       );
122     }
123     return (
124       <form onSubmit={onSubmit}>
125         <Grid>
126           <GridCell span={12} className="form-row">
127             Nesta página, você consegue cadastrar novas universidades no
                ↪ sistema
128             do Guru da Matrícula. Antes de fazer o cadastro, certifique-se
                ↪ que
129             você possui os dados necessários, como um sistema capaz de
                ↪ localizar e
130             hospedar os dados da universidade desejada no formato aceito
                ↪ pelo Guru
131             da Matrícula, e um servidor capaz de lidar com as requisições
                ↪ que o
132             sistema possa fazer. Note que
133             <b> não é possível apagar uma universidade cadastrada após
                ↪ criada</b>,
134             então certifique-se bem antes de confirmar o cadastro.
135           </GridCell>
136           <GridCell span={12} className="form-row">
137             <TextField
138               name="name"
139               type="text"
140               label="Nome"
141               icon="school"
142               required
143               {...nameField}
144             />
145           </GridCell>
146           <GridCell span={12} className="form-row">
147             <TextField
148               name="acronym"
149               type="text"
150               label="Acrônimo"
151               icon="short_text"
152               required
153               {...acronymField}
```

```
154           />
155         </GridCell>
156         <GridCell span={12} className="form-row">
157           <Typography use="headline5">
158             Dados de Ofertas de Disciplinas:
159           </Typography>
160         </GridCell>
161         <GridCell span={12} className="form-row">
162           <TextField
163             name="about_url_offers"
164             type="url"
165             label="URL sobre a fonte dos dados"
166             icon="link"
167             helpText="Esta URL será compartilhada publicamente e
                ↪ normalmente se refere à alguma página dentro do site da
                ↪ universidade"
168             required
169             {...aboutURLOffersField}
170           />
171         </GridCell>
172         <GridCell span={12} className="form-row">
173           <TextField
174             name="base_url_offers"
175             type="url"
176             label="URL para download dos dados"
177             icon="cloud_download"
178             helpText="Utilize {filename} para referenciar o arquivo a ser
                ↪ baixado"
179             required
180             {...baseURLOffersField}
181           />
182         </GridCell>
183         <GridCell span={12} className="form-row">
184           <TextField
185             name="delete_url_offers"
186             type="url"
187             label="URL para apagar arquivo de dados"
```

```
188             helpText="Esta URL será invocada, com o parâmetro 'filenames',
                ↪ assim que o robô terminar o processamento do respectivo
                ↪ arquivo"
189             icon="delete"
190             required
191             {...deleteURLOffersField}
192           />
193         </GridCell>
194         <GridCell span={12} className="form-row">
195           <Typography use="headline5">
196             Dados de Cursos e Habilitações:
197           </Typography>
198         </GridCell>
199         <GridCell span={12} className="form-row">
200           <TextField
201             name="about_url_habilitations"
202             type="url"
203             label="URL sobre a fonte dos dados"
204             helpText="Esta URL será compartilhada publicamente e
                ↪ normalmente se refere à alguma página dentro do site da
                ↪ universidade"
205             icon="link"
206             required
207             {...aboutURLHabilitationsField}
208           />
209         </GridCell>
210         <GridCell span={12} className="form-row">
211           <TextField
212             name="base_url_habilitations"
213             type="url"
214             label="URL para download dos dados"
215             icon="cloud_download"
216             helpText="Utilize {filename} para referenciar o arquivo a ser
                ↪ baixado"
217             required
218             {...baseURLHabilitationsField}
219           />
220         </GridCell>
221         <GridCell span={12} className="form-row">
```

```
222             <TextField
223               name="delete_url_habilitations"
224               type="url"
225               label="URL para apagar arquivo de dados"
226               helpText="Esta URL será invocada, com o parâmetro 'filenames',
                  ↪  assim que o robô terminar o processamento do respectivo
                  ↪  arquivo"
227               icon="delete"
228               required
229               {...deleteURLHabilitationsField}
230             />
231           </GridCell>
232           <GridCell span={12} className="form-row">
233             <Button type="submit" raised>
234               Cadastrar
235             </Button>
236           </GridCell>
237         </Grid>
238         {error ? <Snackbar open onClose={clearError} message={error} /> :
          ↪  null}
239       </form>
240     );
241   }
```

B.2.41 Pasta src/js/pages/create-university/queries

Arquivo create-university.gql

```
1  mutation createUniversity($university: UniversityInput!) {
2    setUniversity(university: $university) {
3      id
4      acronym
5      name
6      offers {
7        secret
8      }
9      habilitations {
10       secret
11     }
```

```
  }
}
```

B.2.42 Pasta src/js/pages/discipline-offer/components

Arquivo index.tsx

```tsx
import React, { useEffect, Fragment, useCallback } from "react";
import { useSelector, useDispatch } from "react-redux";
import { getQueryString, redirect } from "../../base/store/querystring";
import { Grid, GridCell } from "@rmwc/grid";
import { Typography } from "@rmwc/typography";
import { useQuery } from "@apollo/react-hooks";
import GetDisciplineOffer from "../queries/get-discipline-offer.gql";
import {
  RemoteGetDisciplineOfferQuery,
  RemoteGetDisciplineOfferQueryVariables
} from "../../../graphql";
import Title from "../../base/components/title";
import TeamCard from "../../base/components/cards/team";
import { CalendarProvider } from "../../base/components/calendar";

export type DisciplineOfferDetailsModel = Omit<
  Exclude<RemoteGetDisciplineOfferQuery["disciplineOffer"], undefined>,
  "campus"
> & {
  campus?: Exclude<
    RemoteGetDisciplineOfferQuery["disciplineOffer"],
    undefined
  >["campus"];
};

export type TeamDisciplineOfferDetailsModel =
  ↪ DisciplineOfferDetailsModel["teams"][0];

type DisciplineOfferDetailsProps = {
  disciplineOffer: DisciplineOfferDetailsModel;
  onDetails?: (
    disciplineOffer: DisciplineOfferDetailsModel,
    team: TeamDisciplineOfferDetailsModel
```

```
33    ) => void;
34    onAction?: (
35      disciplineOffer: DisciplineOfferDetailsModel,
36      team: TeamDisciplineOfferDetailsModel
37    ) => void;
38    isEnabled?: (
39      disciplineOffer: DisciplineOfferDetailsModel,
40      team: TeamDisciplineOfferDetailsModel
41    ) => boolean;
42    isSelected?: (
43      disciplineOffer: DisciplineOfferDetailsModel,
44      team: TeamDisciplineOfferDetailsModel
45    ) => boolean;
46  };
47
48  export function DisciplineOfferDetails(props:
    ↪  DisciplineOfferDetailsProps) {
49    let { disciplineOffer, onDetails, onAction, isEnabled, isSelected } =
      ↪  props;
50    let { campus } = disciplineOffer;
51    return (
52      <Grid className="team">
53        <GridCell span={12}>
54          <Typography use="headline3">
55            {disciplineOffer.code} - {disciplineOffer.name}
56          </Typography>
57        </GridCell>
58        <GridCell desktop={2} phone={4} tablet={4}>
59          <Typography use="headline5">Universidade:</Typography>
60        </GridCell>
61        <GridCell desktop={10} phone={4} tablet={4}>
62          <Typography
            ↪  use="body1">{disciplineOffer.university.name}</Typography>
63        </GridCell>
64        <GridCell desktop={2} phone={4} tablet={4}>
65          <Typography use="headline5">Período:</Typography>
66        </GridCell>
67        <GridCell desktop={10} phone={4} tablet={4}>
```

```
68          <Typography
            ↪  use="body1">{disciplineOffer.period.name}</Typography>
69        </GridCell>
70        {!!campus && !!campus.id ? (
71          <Fragment>
72            <GridCell desktop={2} phone={4} tablet={4}>
73              <Typography use="headline5">Campus:</Typography>
74            </GridCell>
75            <GridCell desktop={10} phone={4} tablet={4}>
76              <Typography use="body1">{campus.name}</Typography>
77            </GridCell>
78          </Fragment>
79        ) : null}
80        <GridCell span={12}>
81          <Typography use="headline5">Turmas:</Typography>
82        </GridCell>
83        <CalendarProvider width={344} height={194}>
84          {disciplineOffer.teams.map(team => (
85            <GridCell span={4} key={team.id}>
86              <TeamCard
87                onDetails={
88                  onDetails
89                    ? () => onDetails && onDetails(disciplineOffer, team)
90                    : undefined
91                }
92                onAction={
93                  onAction
94                    ? () => onAction && onAction(disciplineOffer, team)
95                    : undefined
96                }
97                enabled={isEnabled && isEnabled(disciplineOffer, team)}
98                selected={isSelected && isSelected(disciplineOffer, team)}
99                team={{
100                 ...team,
101                 discipline: disciplineOffer,
102                 campus: campus
103               }}
104             />
105           </GridCell>
```

```
        ))}
      </CalendarProvider>
    </Grid>
  );
}

type RemoteDisciplineOfferDetailsProps = Omit<
  DisciplineOfferDetailsProps,
  "disciplineOffer"
> & {
  offerId: string;
  changeTitle?: boolean;
};

export function RemoteDisciplineOfferDetails(
  props: RemoteDisciplineOfferDetailsProps
) {
  let { changeTitle, offerId, ...events } = props;
  let { loading, data, error } = useQuery<
    RemoteGetDisciplineOfferQuery,
    RemoteGetDisciplineOfferQueryVariables
  >(GetDisciplineOffer, {
    variables: {
      offerId
    }
  });
  if (error) {
    return <p>Houve um erro durante o carregamento dos dados.</p>;
  }
  if (loading) {
    return <p>Carregando oferta de disciplina...</p>;
  }
  if (!data || !data.disciplineOffer) {
    return <p>Oferta de disciplina não encontrada</p>;
  }
  let { disciplineOffer } = data;
  let result = (
    <DisciplineOfferDetails disciplineOffer={disciplineOffer} {...events}
    ↪ />
```

```
144   );
145   if (!changeTitle) {
146     return result;
147   }
148   return (
149     <Title title={disciplineOffer.code + " - " + disciplineOffer.name}>
150       {result}
151     </Title>
152   );
153 }
154
155 export default function DisciplineOfferPage() {
156   let queryString = useSelector(getQueryString);
157   let id = queryString["id"];
158   let dispatch = useDispatch();
159   let details = useCallback(
160     function(
161       disciplineOffer: DisciplineOfferDetailsModel,
162       team: TeamDisciplineOfferDetailsModel
163     ) {
164       return dispatch(
165         redirect({
166           url: "/turmas/detalhes",
167           querystring: {
168             offer_id: disciplineOffer.id,
169             team_id: team.id
170           }
171         })
172       );
173     },
174     [dispatch]
175   );
176   if (typeof id !== "string") {
177     return <p>Requisição inválida.</p>;
178   }
179   return (
180     <RemoteDisciplineOfferDetails
181       offerId={id}
182       changeTitle
```

```
183       onDetails={details}
184     />
185   );
186 }
```

B.2.43 Pasta src/js/pages/discipline-offer/queries

Arquivo get-discipline-offer.gql

```
1  query getDisciplineOffer($offerId: ID!) {
2    disciplineOffer(id: $offerId) {
3      id
4      university {
5        id
6        name
7      }
8      period {
9        id
10       name
11     }
12     campus {
13       id
14       name
15     }
16     id
17     code
18     name
19     teams {
20       id
21       code
22       vacancies {
23         offered
24         filled
25       }
26       schedules {
27         start {
28           hour
29           minute
30         }
31         end {
```

```
            hour
            minute
          }
          dayOfWeek
        }
      }
    }
}
```

B.2.44 Pasta src/js/pages/discipline/components

Arquivo index.tsx

```tsx
import React, { useEffect, Fragment, useCallback } from "react";
import { useSelector, useDispatch } from "react-redux";
import { getQueryString, pushQueryString } from
  "../../base/store/querystring";
import { Grid, GridCell } from "@rmwc/grid";
import { Typography } from "@rmwc/typography";
import { useLazyQuery, useQuery } from "@apollo/react-hooks";
import getDiscipline from "../queries/get-discipline.gql";
import {
  RemoteGetDisciplineQuery,
  RemoteGetDisciplineQueryVariables,
  RemoteDisciplineSummaryFragment
} from "../../../graphql";
import RelatedDisciplines from "./related";
import Title from "../../base/components/title";
import { Table } from "../../base/components/table";

type DisciplineDetailsProps = {
  useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
    void;
  discipline: Exclude<RemoteGetDisciplineQuery["discipline"], undefined>;
};

function DisciplineTables(props: DisciplineDetailsProps) {
  let { discipline } = props;
  if (!discipline || !discipline.tables || !discipline.tables.length) {
    return null;
```

```
26    }
27    return (
28      <Fragment>
29        <GridCell span={12}>
30          <Typography use="headline5">Outras informações:</Typography>
31        </GridCell>
32        {discipline.tables.map(table => (
33          <GridCell span={12}>
34            <Table table={table} />
35          </GridCell>
36        ))}
37      </Fragment>
38    );
39  }
40
41  export function DisciplineDetails(props: DisciplineDetailsProps) {
42    let { discipline, useDetails } = props;
43    return (
44      <Grid className="discipline">
45        <GridCell span={12}>
46          <Typography use="headline3">
47            {discipline.code} - {discipline.name}
48          </Typography>
49        </GridCell>
50        <GridCell span={12}>
51          <Typography use="body1">{discipline.description}</Typography>
52        </GridCell>
53        <GridCell desktop={2} phone={4} tablet={4}>
54          <Typography use="headline5">Tipo:</Typography>
55        </GridCell>
56        <GridCell desktop={10} phone={4} tablet={4}>
57          <Typography use="body1">{discipline.type}</Typography>
58        </GridCell>
59        <GridCell desktop={2} phone={4} tablet={4}>
60          <Typography use="headline5">Universidade:</Typography>
61        </GridCell>
62        <GridCell desktop={10} phone={4} tablet={4}>
63          <Typography use="body1">{discipline.university.name}</Typography>
```

```
64          </GridCell>
65          <GridCell desktop={2} phone={4} tablet={4}>
66            <Typography use="headline5">Curso:</Typography>
67          </GridCell>
68          <GridCell desktop={10} phone={4} tablet={4}>
69            <Typography use="body1">{discipline.course.name}</Typography>
70          </GridCell>
71          <GridCell desktop={2} phone={4} tablet={4}>
72            <Typography use="headline5">Habilitação:</Typography>
73          </GridCell>
74          <GridCell desktop={10} phone={4} tablet={4}>
75            <Typography use="body1">{discipline.habilitation.name}
              ↪  </Typography>
76          </GridCell>
77          <DisciplineTables {...props} />
78          <GridCell span={12}>
79            <Typography use="headline4">Disciplinas
              ↪  Relacionadas:</Typography>
80          </GridCell>
81          <GridCell span={12}>
82            <RelatedDisciplines
83              relatedDisciplines={discipline.related}
84              disciplines={discipline.relatedDisciplines}
85              useDetails={useDetails}
86            ></RelatedDisciplines>
87          </GridCell>
88        </Grid>
89    );
90  }
91
92  type RemoteDisciplineDetailsProps = {
93    useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
      ↪  void;
94    disciplineId: string;
95    changeTitle?: boolean;
96  };
97
```

```
98  export function RemoteDisciplineDetails(props:
    ↪  RemoteDisciplineDetailsProps) {
99    let { changeTitle, useDetails, ...variables } = props;
100   let { loading, data, error } = useQuery<
101     RemoteGetDisciplineQuery,
102     RemoteGetDisciplineQueryVariables
103   >(getDiscipline, {
104     variables
105   });
106   if (error) {
107     return <p>Houve um erro durante o carregamento dos dados.</p>;
108   }
109   if (loading) {
110     return <p>Carregando disciplina...</p>;
111   }
112   if (!data || !data.discipline) {
113     return <p>Disciplina não encontrada</p>;
114   }
115   let { discipline } = data;
116   let result = (
117     <DisciplineDetails discipline={discipline} useDetails={useDetails} />
118   );
119   if (!changeTitle) {
120     return result;
121   }
122   return (
123     <Title title={discipline.code + " - " +
      ↪  discipline.name}>{result}</Title>
124   );
125 }
126
127 export default function DisciplinePage() {
128   let dispatch = useDispatch();
129   let useDetails = useCallback(
130     function({ id }: RemoteDisciplineSummaryFragment) {
131       return function() {
132         dispatch(
133           pushQueryString({
134             id
```

```
135           })
136         );
137       };
138     },
139     [dispatch]
140   );
141   let queryString = useSelector(getQueryString);
142   let id = queryString["id"];
143   if (typeof id !== "string") {
144     return <p>Requisição inválida.</p>;
145   }
146   return (
147     <RemoteDisciplineDetails
148       disciplineId={id}
149       changeTitle
150       useDetails={useDetails}
151     />
152   );
153 }
```

Arquivo related.tsx

```
1  import {
2    RemoteRelatedFragment,
3    RemoteDisciplineRelatedType,
4    RemoteDisciplineRelatedItem,
5    RemoteDisciplineRelatedItemType,
6    RemoteDisciplineSummaryFragment
7  } from "../../../graphql";
8  import React, { Fragment, useCallback } from "react";
9  import { GridInner, GridCell } from "@rmwc/grid";
10 import {
11   Card,
12   CardPrimaryAction,
13   CardActions,
14   CardActionButtons,
15   CardActionButton
16 } from "@rmwc/card";
17 import { Typography } from "@rmwc/typography";
```

```
18 import { ListDivider } from "@rmwc/list";
19
20 export type RelatedItemProps = {
21   item: RemoteDisciplineRelatedItem;
22   disciplines: RemoteDisciplineSummaryFragment[];
23   useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
     ↪ void;
24 };
25
26 export function RelatedText({ item }: RelatedItemProps) {
27   return (
28     <CardPrimaryAction>
29       <div style={{ padding: "1rem" }}>
30         <Typography use="body1" tag="p"
           ↪ theme="textSecondaryOnBackground">
31           {item.description}
32         </Typography>
33       </div>
34     </CardPrimaryAction>
35   );
36 }
37
38 export function RelatedGenericDiscipline(props: RelatedItemProps) {
39   let { item, disciplines, useDetails } = props;
40   let rows = disciplines.filter(function(row) {
41     return row.genericID === item.value;
42   });
43   switch (rows.length) {
44     case 0:
45       return (
46         <CardPrimaryAction>
47           <div style={{ padding: "1rem" }}>
48             <Typography use="headline5">{item.description}</Typography>
49             <br />
50             <Typography use="body2">
51               Esta disciplina não foi encontrada no sistema.
52             </Typography>
53           </div>
54         </CardPrimaryAction>
```

```
55        );
56      case 1:
57        return (
58          <RelatedDisciplineCard discipline={rows[0]}
            ↪  useDetails={useDetails} />
59        );
60      default:
61        return (
62          <CardPrimaryAction>
63            <div style={{ padding: "1rem" }}>
64              <Typography use="headline5">{item.description}</Typography>
65              <br />
66              <Typography use="body2">
67                Esta disciplina se refere a {rows.length} registros no
                  ↪  sistema.
68              </Typography>
69            </div>
70          </CardPrimaryAction>
71        );
72    }
73  }
74
75  type RelatedDisciplineCardProps = {
76    discipline: RemoteDisciplineSummaryFragment;
77    useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
      ↪  void;
78  };
79
80  function RelatedDisciplineCard(props: RelatedDisciplineCardProps) {
81    let { discipline, useDetails } = props;
82    let details = useDetails(discipline);
83
84    return (
85      <CardPrimaryAction>
86        <div style={{ padding: "1rem" }}>
87          <Typography use="headline6">
88            {discipline.code} - {discipline.name}
89          </Typography>
90          <br />
```

```
91          <Typography use="subtitle1">{discipline.type}</Typography>
92          <br />
93          <Typography use="body2">{discipline.description}</Typography>
94        </div>
95        <CardActions>
96          <CardActionButtons>
97            <CardActionButton outlined trailingIcon="details"
              ↪  onClick={details}>
98              Detalhes
99            </CardActionButton>
100         </CardActionButtons>
101       </CardActions>
102     </CardPrimaryAction>
103   );
104 }
105
106 export function RelatedDiscipline(props: RelatedItemProps) {
107   let { item, disciplines, useDetails } = props;
108   let discipline = disciplines.find(function(row) {
109     return row.id === item.value;
110   });
111   if (!discipline) {
112     return (
113       <CardPrimaryAction>
114         <div style={{ padding: "1rem" }}>
115           <Typography use="headline5">
116             Erro durante o carregamento da disciplina
                ↪  "{item.description}"..
117           </Typography>
118         </div>
119       </CardPrimaryAction>
120     );
121   }
122   return (
123     <RelatedDisciplineCard discipline={discipline} useDetails={useDetails}
      ↪  />
124   );
125 }
126
```

```
127 export function RelatedItem(props: RelatedItemProps) {
128   switch (props.item.type) {
129     case RemoteDisciplineRelatedItemType.DisciplineGenericId:
130       return <RelatedGenericDiscipline {...props} />;
131     case RemoteDisciplineRelatedItemType.DisciplineId:
132       return <RelatedDiscipline {...props} />;
133     case RemoteDisciplineRelatedItemType.Text:
134       return <RelatedText {...props} />;
135   }
136 }
137
138 export type RelatedProps = {
139   relatedDiscipline: RemoteRelatedFragment;
140   disciplines: RemoteDisciplineSummaryFragment[];
141   useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
    ↪  void;
142 };
143
144 export function Related(props: RelatedProps) {
145   let { relatedDiscipline, disciplines, useDetails } = props;
146   let type: string = "Desconhecido";
147   switch (relatedDiscipline.type) {
148     case RemoteDisciplineRelatedType.Equivalent:
149       type = "Equivalente";
150       break;
151     case RemoteDisciplineRelatedType.Prerequisite:
152       type = "Pré-Requisito";
153       break;
154     case RemoteDisciplineRelatedType.Set:
155       type = "Conjunto";
156       break;
157   }
158   return (
159     <Card outlined>
160       <Typography
161         use="headline5"
162         tag="div"
163         style={{ padding: "0.5rem 0.5rem" }}
164         theme="textPrimaryOnBackground"
```

```
165         >
166           {relatedDiscipline.name}
167         </Typography>
168         <Typography
169           use="caption"
170           style={{ padding: "0.5rem 0.5rem" }}
171           theme="textSecondaryOnBackground"
172         >
173           Tipo: {type}
174         </Typography>
175         <ListDivider />
176         {relatedDiscipline.items.map(item => (
177           <Fragment key={item.value + "-" + item.type + "-" +
            ↪ item.description}>
178             <RelatedItem
179               item={item}
180               disciplines={disciplines}
181               useDetails={useDetails}
182             />
183             <ListDivider />
184           </Fragment>
185         ))}
186       </Card>
187     );
188   }
189
190   export type RelatedDisciplinesProps = {
191     relatedDisciplines: RemoteRelatedFragment[];
192     disciplines: RemoteDisciplineSummaryFragment[];
193     useDetails: (discipline: RemoteDisciplineSummaryFragment) => () =>
      ↪ void;
194   };
195
196   export default function RelatedDisciplines(props:
    ↪ RelatedDisciplinesProps) {
197     let { relatedDisciplines, disciplines, useDetails } = props;
198     return (
199       <GridInner>
200         {relatedDisciplines.map((item, index) => (
```

```
201         <GridCell span={4} key={index}>
202           <Related
203             relatedDiscipline={item}
204             disciplines={disciplines}
205             useDetails={useDetails}
206           />
207         </GridCell>
208       ))}
209     </GridInner>
210   );
211 }
```

B.2.45 Pasta src/js/pages/discipline/queries

Arquivo get-discipline.gql

```
1  #import "./related.gql"
2  #import "../../base/queries/table.gql"
3  query getDiscipline($disciplineId: ID!) {
4    discipline(id: $disciplineId) {
5      id
6      name
7      university {
8        id
9        name
10     }
11     course {
12       id
13       name
14     }
15     habilitation {
16       id
17       name
18     }
19     step {
20       id
21       name
22     }
23     genericID
24     code
```

```
    name
    description
    type
    tables {
      ...table
    }
    related {
      ...related
    }
    relatedDisciplines {
      ...disciplineSummary
    }
  }
}
```

Arquivo related.gql

```
fragment related on DisciplineRelated {
  name
  type
  items {
    type
    value
    description
  }
}

fragment disciplineSummary on Discipline {
  id
  genericID
  code
  name
  type
  description
}
```

B.2.46 Pasta src/js/pages/edit-university/components

Arquivo index.tsx

```
import React, { Fragment } from "react";
import {
  useSubmit,
  useTextField,
  useCheckboxField
} from "../../base/components/form";
import { Button } from "@rmwc/button";
import { Grid, GridCell } from "@rmwc/grid";
import { TextField } from "@rmwc/textfield";
import { Snackbar } from "@rmwc/snackbar";
import { Switch } from "@rmwc/switch";
import { useRedirectIfNotAuthenticated } from
↪  "../../base/components/auth";
import { Typography } from "@rmwc/typography";
import { useApolloClient, useQuery } from "@apollo/react-hooks";
import ApolloClient from "apollo-client";
import {
  getUniversity,
  updateUniversity
} from "../queries/edit-university.gql";
import {
  RemoteCreateUniversityMutation,
  RemoteCreateUniversityMutationVariables,
  RemoteUniversityInput,
  RemoteGetUniversityQuery,
  RemoteGetUniversityQueryVariables
} from "../../../graphql";
import { useSelector } from "react-redux";
import { getQueryString } from "../../../base/store/querystring";

async function submit<CacheShape>(
  client: ApolloClient<CacheShape>,
  university: RemoteUniversityInput
): Promise<RemoteCreateUniversityMutation> {
  let result = await client.mutate<
    RemoteCreateUniversityMutation,
    RemoteCreateUniversityMutationVariables
  >({
    mutation: updateUniversity,
```

```
39      variables: {
40        university
41      }
42    });
43    if (!result.data) {
44      return Promise.reject("Erro inesperado durante o cadastro da
        ↪ universidade");
45    }
46    return result.data;
47  }
48
49  type EditUniversityFormProps = { id: string; data:
    ↪ RemoteGetUniversityQuery };
50  function EditUniversityForm({ id, data }: EditUniversityFormProps) {
51    const { university } = data;
52    const [acronym, acronymField] = useTextField(university?.acronym);
53    const [name, nameField] = useTextField(university?.name);
54    const [publicValue, publicField] =
      ↪ useCheckboxField(university?.public);
55    const [baseURLOffers, baseURLOffersField] = useTextField(
56      university?.offers.baseUrl
57    );
58    const [deleteURLOffers, deleteURLOffersField] = useTextField(
59      university?.offers.deleteUrl
60    );
61    const [aboutURLOffers, aboutURLOffersField] = useTextField(
62      university?.offers.aboutUrl
63    );
64    const [generateSecretOffers, generateSecretOffersField] =
      ↪ useCheckboxField();
65    const [baseURLHabilitations, baseURLHabilitationsField] = useTextField(
66      university?.habilitations.baseUrl
67    );
68    const [deleteURLHabilitations, deleteURLHabilitationsField] =
      ↪ useTextField(
69      university?.habilitations.deleteUrl
70    );
71    const [aboutURLHabilitations, aboutURLHabilitationsField] =
      ↪ useTextField(
```

```
72      university?.habilitations.aboutUrl
73    );
74    const [
75      generateSecretHabilitations,
76      generateSecretHabilitationsField
77    ] = useCheckboxField();
78    const client = useApolloClient();
79    const { onSubmit, submitted, error, clearError } = useSubmit(() =>
80      submit(client, {
81        id,
82        acronym,
83        name,
84        public: publicValue,
85        offers: {
86          baseUrl: baseURLOffers,
87          aboutUrl: aboutURLOffers,
88          deleteUrl: deleteURLOffers,
89          generateSecret: generateSecretOffers
90        },
91        habilitations: {
92          baseUrl: baseURLHabilitations,
93          aboutUrl: aboutURLHabilitations,
94          deleteUrl: deleteURLHabilitations,
95          generateSecret: generateSecretHabilitations
96        }
97      })
98    );
99    if (submitted) {
100     return (
101       <Grid>
102         <GridCell span={12}>Universidade editada!</GridCell>
103         {generateSecretOffers ? (
104           <Fragment>
105             <GridCell span={6}>O segredo para as ofertas de turmas
                ↪ é:</GridCell>
106             <GridCell span={6}>
107               <TextField
108                 value={submitted.setUniversity.offers.secret}
109                 readOnly
```

```
            />
          </GridCell>
        </Fragment>
      ) : null}
      {generateSecretHabilitations ? (
        <Fragment>
          <GridCell span={6}>
            O segredo para os dados de habilitações é:
          </GridCell>
          <GridCell span={6}>
            <TextField
              value={submitted.setUniversity.habilitations.secret}
              readOnly
            />
          </GridCell>
        </Fragment>
      ) : null}
    </Grid>
  );
}
return (
  <form onSubmit={onSubmit}>
    <Grid>
      <GridCell span={12} className="form-row">
        Nesta página, você consegue cadastrar novas páginas no sistema
        ↪ do Guru
        da Matrícula. Antes de fazer o cadastro, certifique-se que você
        ↪ possui
        os dados necessários, como um sistema capaz de localizar e
        ↪ hospedar os
        dados da universidade desejada no formato aceito pelo Guru da
        Matrícula, e um servidor capaz de lidar com as requisições que
        ↪ o
        sistema possa fazer.
      </GridCell>
      <GridCell span={12} className="form-row">
        <TextField
          name="name"
          type="text"
```

```
145             label="Nome"
146             icon="school"
147             required
148             {...nameField}
149           />
150         </GridCell>
151         <GridCell span={12} className="form-row">
152           <TextField
153             name="acronym"
154             type="text"
155             label="Acrônimo"
156             icon="short_text"
157             required
158             {...acronymField}
159           />
160         </GridCell>
161         <GridCell span={12} className="form-row">
162           <Switch name="public" label="Publica" {...publicField} />
163         </GridCell>
164         <GridCell span={12} className="form-row">
165           <Typography use="headline5">
166             Dados de Ofertas de Disciplinas:
167           </Typography>
168         </GridCell>
169         <GridCell span={12} className="form-row">
170           <TextField
171             name="about_url_offers"
172             type="url"
173             label="URL sobre a fonte dos dados"
174             icon="link"
175             helpText="Esta URL será compartilhada publicamente e
                ↪ normalmente se refere à alguma página dentro do site da
                ↪ universidade"
176             required
177             {...aboutURLOffersField}
178           />
179         </GridCell>
180         <GridCell span={12} className="form-row">
181           <TextField
```

```
182             name="base_url_offers"
183             type="url"
184             label="URL para download dos dados"
185             icon="cloud_download"
186             helpText="Utilize {filename} para referenciar o arquivo a ser
                ↪  baixado"
187             required
188             {...baseURLOffersField}
189           />
190         </GridCell>
191         <GridCell span={12} className="form-row">
192           <TextField
193             name="delete_url_offers"
194             type="url"
195             label="URL para apagar arquivo de dados"
196             helpText="Esta URL será invocada, com o parâmetro 'filenames',
                ↪  assim que o robô terminar o processamento do respectivo
                ↪  arquivo"
197             icon="delete"
198             required
199             {...deleteURLOffersField}
200           />
201         </GridCell>
202         <GridCell span={12} className="form-row">
203           <Switch
204             name="generate_secret_offers"
205             label="Gerar novo segredo?"
206             {...generateSecretOffersField}
207           />
208         </GridCell>
209         <GridCell span={12} className="form-row">
210           <Typography use="headline5">
211             Dados de Cursos e Habilitações:
212           </Typography>
213         </GridCell>
214         <GridCell span={12} className="form-row">
215           <TextField
216             name="about_url_habilitations"
217             type="url"
```

```
218             label="URL sobre a fonte dos dados"
219             helpText="Esta URL será compartilhada publicamente e
                ↪ normalmente se refere à alguma página dentro do site da
                ↪ universidade"
220             icon="link"
221             required
222             {...aboutURLHabilitationsField}
223           />
224         </GridCell>
225         <GridCell span={12} className="form-row">
226           <TextField
227             name="base_url_habilitations"
228             type="url"
229             label="URL para download dos dados"
230             icon="cloud_download"
231             helpText="Utilize {filename} para referenciar o arquivo a ser
                ↪ baixado"
232             required
233             {...baseURLHabilitationsField}
234           />
235         </GridCell>
236         <GridCell span={12} className="form-row">
237           <TextField
238             name="delete_url_habilitations"
239             type="url"
240             label="URL para apagar arquivo de dados"
241             helpText="Esta URL será invocada, com o parâmetro 'filenames',
                ↪ assim que o robô terminar o processamento do respectivo
                ↪ arquivo"
242             icon="delete"
243             required
244             {...deleteURLHabilitationsField}
245           />
246         </GridCell>
247         <GridCell span={12} className="form-row">
248           <Switch
249             name="generate_secret_habilitations"
250             label="Gerar novo segredo?"
251             {...generateSecretHabilitationsField}
```

```
252             />
253           </GridCell>
254           <GridCell span={12} className="form-row">
255             <Button type="submit" raised>
256               Editar
257             </Button>
258           </GridCell>
259         </Grid>
260         {error ? <Snackbar open onClose={clearError} message={error} /> :
            ↪ null}
261       </form>
262     );
263   }
264   export default function EditUniversity() {
265     useRedirectIfNotAuthenticated();
266     const querystring = useSelector(getQueryString);
267     let id: string = "";
268     if (querystring["id"] && !Array.isArray(querystring["id"])) {
269       id = querystring["id"];
270     }
271     if (!id) {
272       return (
273         <p>
274           Requisição inválida. É necessário um identificador de
              ↪ universidade para
275           editar.
276         </p>
277       );
278     }
279     let { data, loading, error } = useQuery<
280       RemoteGetUniversityQuery,
281       RemoteGetUniversityQueryVariables
282     >(getUniversity, {
283       variables: {
284         id
285       }
286     });
287     if (error) {
288       return <p>Houve um erro durante o carregamento da universidade.</p>;
```

```
289   }
290   if (loading) {
291     return <p>Carregando...</p>;
292   }
293   if (!data || !data.university) {
294     return <p>Universidade não encontrada</p>;
295   }
296   return <EditUniversityForm id={id} data={data} />;
297 }
```

B.2.47 Pasta src/js/pages/edit-university/queries

Arquivo edit-university.gql

```
1  query getUniversity($id: ID!) {
2    university(id: $id) {
3      acronym
4      name
5      public
6      offers {
7        baseUrl
8        deleteUrl
9        aboutUrl
10     }
11     habilitations {
12       baseUrl
13       deleteUrl
14       aboutUrl
15     }
16   }
17 }
18
19 mutation updateUniversity($university: UniversityInput!) {
20   setUniversity(university: $university) {
21     id
22     acronym
23     name
24     public
25     offers {
26       secret
```

```
27      }
28      habilitations {
29        secret
30      }
31    }
32  }
```

B.2.48 Pasta src/js/pages/help/components

Arquivo index.tsx

```
1  import React, { useEffect, Fragment } from "react";
2  import Title from "../../base/components/title";
3  import { Grid, GridCell } from "@rmwc/grid";
4  import { Typography } from "@rmwc/typography";
5
6  export default function HelpPage() {
7    return (
8      <Grid>
9        <GridCell span={12}>
10         <Typography use="overline">Ajuda</Typography>
11       </GridCell>
12       <GridCell span={12}>
13         <Typography use="headline3">Perguntas Frequentes</Typography>
14       </GridCell>
15     </Grid>
16   );
17 }
```

B.2.49 Pasta src/js/pages/login/components

Arquivo index.tsx

```
1  import React, { useEffect } from "react";
2  import { useSubmit, useTextField } from "../../base/components/form";
3  import { Button } from "@rmwc/button";
4  import { BASE_BACKEND_URL } from "../../../config";
5  import { Grid, GridInner, GridCell } from "@rmwc/grid";
6  import { Link } from "react-router-dom";
7  import { TextField } from "@rmwc/textfield";
```

```
8  import { Snackbar } from "@rmwc/snackbar";
9  import { useSelector, useDispatch } from "react-redux";
10 import { getQueryString, getLinkSelector } from
   ↪  "../../base/store/querystring";
11 import { update as updateAction } from "../../base/store/auth";
12 import { useRedirectIfAuthenticated } from "../../base/components/auth";
13
14 const callbackMessage = "oauth2_callback_ok";
15 const answerMessage = "oauth2_callback_answer";
16 const oauth2ParamName = "callback";
17 const oauth2ParamValue = "1";
18
19 interface Fields {
20   email: string;
21   password: string;
22 }
23
24 async function submit(
25   csrfToken: string | null,
26   fields: Fields
27 ): Promise<boolean> {
28   let error: string | null =
29     "Erro durante autenticação. Verifique seu e-mail e senha.";
30   if (csrfToken) {
31     try {
32       let response: Response = await
         ↪  fetch(`${BASE_BACKEND_URL}/auth/login`, {
33         body: JSON.stringify(fields),
34         method: "POST",
35         credentials: "include",
36         headers: {
37           "X-CSRF-Token": csrfToken
38         }
39       });
40       if (response.ok) {
41         interface APIResponse {
42           status: "success" | "failure";
43           error?: string;
44         }
```

```
45          var data: APIResponse = await response.json();
46          if (data.status === "success") {
47            error = null;
48          }
49        }
50      } catch (e) {
51        error =
52          "Erro desconhecido ao realizar autenticação. Tente novamente mais
            ↪ tarde.";
53      }
54    } else {
55      error = "Erro interno ao enviar requisição. Tente recarregar a
        ↪ página.";
56    }
57    return error === null || Promise.reject(error);
58  }
59
60  function loginOAuth2(
61    service: "facebook" | "google",
62    showError: (error: string) => void,
63    update: () => void,
64    redirectURL: string
65  ) {
66    try {
67      let handler = window.open(
68        `${BASE_BACKEND_URL}/auth/oauth2/${service}?redir=${redirectURL}`
69      );
70      if (handler) {
71        let receivedMessage = false;
72        let closeTimeout = setInterval(() => {
73          if (!handler || !handler.closed || receivedMessage) {
74            return;
75          }
76          showError(
77            "Erro: A janela fechou sem o processo ter sido finalizado.
              ↪ Tente novamente, por favor."
78          );
79          window.removeEventListener("message", message);
80          clearInterval(closeTimeout);
```

```
81        }, 1000);
82        let message = function(ev: MessageEvent) {
83          if (
84            ev.source === null ||
85            handler === null ||
86            !Object.is(ev.source, handler) ||
87            ev.data !== callbackMessage
88          ) {
89            return;
90          }
91          receivedMessage = true;
92          clearInterval(closeTimeout);
93          update();
94          handler.postMessage(answerMessage, ev.origin);
95          window.removeEventListener("message", message);
96        };
97        window.addEventListener("message", message);
98        handler.focus();
99      } else {
100       showError("Não foi possível abrir a janela.");
101     }
102   } catch (e) {
103     showError("Erro inesperado aconteceu. Tente recarregar a página.");
104   }
105 }
106
107 export default function LoginHook() {
108   let [email, emailField] = useTextField();
109   let [password, passwordField] = useTextField();
110   useRedirectIfAuthenticated();
111   let querystring = useSelector(getQueryString);
112   if (querystring[oauth2ParamName] === oauth2ParamValue && window.opener)
      ↪ {
113     window.addEventListener("message", function(ev: MessageEvent) {
114       if (ev.source !== window.opener || ev.data !== answerMessage) {
115         return;
116       }
117       window.close();
118     });
```

```
    window.opener.postMessage(callbackMessage, window.location.origin);
    return <p>Você pode fechar esta janela agora.</p>;
  }
  let { showError, onSubmit, submitted, error, clearError } = useSubmit(
    (_, csrfToken) => submit(csrfToken, { email, password }),
    querystring["redir"]
      ? "Você precisa estar autenticado para acessar esta página."
      : null
  );
  let dispatch = useDispatch();
  let update = () => dispatch(updateAction());
  let redirectURL = useSelector(
    getLinkSelector({ [oauth2ParamName]: oauth2ParamValue })
  );
  redirectURL = encodeURIComponent(window.location.origin + redirectURL);
  let facebook = () => loginOAuth2("facebook", showError, update,
  ↪  redirectURL);
  let google = () => loginOAuth2("google", showError, update,
  ↪  redirectURL);
  useEffect(
    function() {
      if (!submitted) {
        return;
      }
      dispatch(updateAction());
    },
    [dispatch, submitted]
  );
  if (submitted) {
    return <p>Fazendo login...</p>;
  }
  return (
    <form onSubmit={onSubmit}>
      <Grid>
        <GridCell span={12} className="form-row">
          Você pode se conectar usando:
        </GridCell>
        <GridCell span={12} className="form-row">
          <GridInner>
```

```
156             <GridCell span={2}>
157               <Button onClick={facebook} raised>
158                 Facebook
159               </Button>
160             </GridCell>
161             <GridCell span={2}>
162               <Button onClick={google} raised>
163                 Google
164               </Button>
165             </GridCell>
166           </GridInner>
167         </GridCell>
168         <GridCell span={12} className="form-row">
169           Ou com a sua conta do Guru da Matrícula:
170         </GridCell>
171         <GridCell span={12} className="form-row">
172           <TextField
173             name="email"
174             type="email"
175             label="E-mail"
176             autoComplete="email"
177             icon="email"
178             required
179             {...emailField}
180           />
181         </GridCell>
182         <GridCell span={12} className="form-row">
183           <TextField
184             name="password"
185             type="password"
186             label="Senha"
187             autoComplete="current-password"
188             icon="lock"
189             required
190             {...passwordField}
191           />
192         </GridCell>
193         <GridCell span={12} className="form-row">
194           <Button type="submit" raised>
```

```
            Entrar
          </Button>
        </GridCell>
        <GridCell span={12} className="form-row">
          Não tem cadastro?
          <Link to="/cadastro">Crie sua conta gratuitamente.</Link>
        </GridCell>
        <GridCell span={12} className="form-row">
          Esqueceu sua senha?
          <Link to="/recuperar-senha">Clique aqui para recuperar.</Link>
        </GridCell>
      </Grid>
      {error ? <Snackbar open onClose={clearError} message={error} /> :
      ↪  null}
    </form>
  );
}
```

B.2.50 Pasta src/js/pages/offline/components

Arquivo index.tsx

```
import React, { Fragment } from "react";
import { useQuery } from "@apollo/react-hooks";
import { getOfflineData } from "../queries/get-offline-data.gql";
import { Grid, GridCell } from "@rmwc/grid";
import {
  DataTable,
  DataTableHead,
  DataTableHeadCell,
  DataTableBody,
  DataTableRow,
  DataTableCell,
  DataTableContent
} from "@rmwc/data-table";
import { LinearProgress } from "@rmwc/linear-progress";
import {
  RemoteGetOfflineDataQuery,
  RemoteGetOfflineDataQueryVariables
```

```
} from "../../../graphql";
import { Switch } from "@rmwc/switch";

export default function OfflinePage() {
  let { data, loading, error } = useQuery<
    RemoteGetOfflineDataQuery,
    RemoteGetOfflineDataQueryVariables
  >(getOfflineData);
  if (loading) {
    return (
      <Grid>
        <GridCell span={12}>
          <LinearProgress />
        </GridCell>
        <GridCell span={12}>Carregando dados..</GridCell>
      </Grid>
    );
  }
  if (error || !data) {
    return (
      <Grid>
        <GridCell span={12}>Erro ao carregar os dados.</GridCell>
      </Grid>
    );
  }
  return (
    <Grid>
      <GridCell span={12}>
        No painel abaixo é possível configurar que dados o sistema vai
        ↪ baixar e
        manter atualizado no seu dispositivo. Observe que, quanto mais
        ↪ opções
        você marcar, mais espaço de armazenamento será consumido no seu
        dispositivo, portanto, recomendamos que selecione, abaixo, apenas
        ↪ os
        dados que são realmente importantes para você.
      </GridCell>
      <GridCell span={12}>
        <DataTable stickyRows={1}>
```

```
54            <DataTableContent>
55              <DataTableHead>
56                <DataTableRow>
57                  <DataTableHeadCell>Universidade</DataTableHeadCell>
58                  <DataTableHeadCell>Período</DataTableHeadCell>
59                  <DataTableHeadCell>Curso</DataTableHeadCell>
60                  <DataTableHeadCell>Estado Atual</DataTableHeadCell>
61                </DataTableRow>
62              </DataTableHead>
63              <DataTableBody>
64                {data.universities.map(university => (
65                  <Fragment key={university.id}>
66                    {university.periods.map(period => (
67                      <Fragment key={period.id}>
68                        {university.courses.map(course => (
69                          <DataTableRow key={course.id}>
70
                              ↪  <DataTableCell>{university.acronym}</DataTableCell>
71                            <DataTableCell>{period.name}</DataTableCell>
72                            <DataTableCell>{course.name}</DataTableCell>
73                            <DataTableCell>
74                              <Switch />
75                            </DataTableCell>
76                          </DataTableRow>
77                        ))}
78                      </Fragment>
79                    ))}
80                  </Fragment>
81                ))}
82              </DataTableBody>
83            </DataTableContent>
84          </DataTable>
85        </GridCell>
86      </Grid>
87    );
88  }
```

B.2.51 Pasta src/js/pages/offline/queries

Arquivo get-offline-data.gql

```
query getOfflineData {
  universities {
    id
    acronym
    name
    periods {
      id
      name
    }
    courses {
      id
      name
    }
  }
}
```

B.2.52 Pasta src/js/pages/password-recovery/components

Arquivo index.tsx

```tsx
import React from "react";
import { useTextField, useSubmit } from "../../base/components/form";
import { Button } from "@rmwc/button";
import { BASE_BACKEND_URL } from "../../../config";
import { Grid, GridCell } from "@rmwc/grid";
import { Link } from "react-router-dom";
import { TextField } from "@rmwc/textfield";
import { Snackbar } from "@rmwc/snackbar";

async function submit(
  csrfToken: string | null,
  email: string
): Promise<boolean> {
  let error: string | null =
    "Erro durante a recuperação da conta. Verifique o e-mail informado.";
  if (csrfToken) {
```

```
17      try {
18        let response: Response = await
          ↪ fetch(`${BASE_BACKEND_URL}/auth/recover`, {
19          body: JSON.stringify({ email }),
20          method: "POST",
21          credentials: "include",
22          headers: {
23            "X-CSRF-Token": csrfToken
24          }
25        });
26        if (response.status === 200) {
27          interface APIResponse {
28            status: "success" | "failure";
29            error?: string;
30          }
31          var data: APIResponse = await response.json();
32          if (data.status === "success") {
33            error = null;
34          }
35        }
36      } catch (e) {
37        error =
38          "Erro desconhecido ao realizar autenticação. Tente novamente mais
            ↪ tarde.";
39      }
40    } else {
41      error = "Erro interno ao enviar requisição. Tente recarregar a
        ↪ página.";
42    }
43    return error === null || Promise.reject(error);
44  }
45
46  export default function PasswordRecoveryHook() {
47    let [email, emailField] = useTextField();
48    let { submitted, error, clearError, onSubmit } = useSubmit((evt,
      ↪ csrfToken) =>
49      submit(csrfToken, email)
50    );
51    if (submitted) {
```

```
52      return (
53        <p>
54          Um e-mail foi enviado com um link para iniciar o processo de
            ↪  recuperação
55          da conta. Por favor, cheque sua caixa de entrada.
56        </p>
57      );
58    }
59    return (
60      <form onSubmit={onSubmit}>
61        <Grid>
62          <GridCell span={12} className="form-row">
63            Informe o seu e-mail abaixo para recuperar sua senha:
64          </GridCell>
65          <GridCell span={12} className="form-row">
66            <TextField
67              name="email"
68              type="email"
69              label="E-mail"
70              autoComplete="email"
71              icon="email"
72              required
73              {...emailField}
74            />
75          </GridCell>
76          <GridCell span={12} className="form-row">
77            <Button type="submit" raised>
78              Recuperar
79            </Button>
80          </GridCell>
81          <GridCell span={12} className="form-row">
82            Não tem cadastro?
83            <Link to="/cadastro">Crie sua conta gratuitamente.</Link>
84          </GridCell>
85          <GridCell span={12} className="form-row">
86            Lembrou sua senha?
87            <Link to="/entrar">Faça login.</Link>
88          </GridCell>
89        </Grid>
```

```
90        {error ? <Snackbar open onClose={clearError} message={error} /> :
          ↪  null}
91      </form>
92    );
93  }
```

B.2.53 Pasta src/js/pages/plan-details/components

Arquivo container.tsx

```
1  import React, { ReactNode, Fragment } from "react";
2  import PlanTopBar from "./top-bar";
3  import { TopBarContent } from "../../base/components/partials/top-bar";
4
5  type Props = {
6    children: ReactNode;
7  };
8
9  export default function PlanContainer(props: Props) {
10   return (
11     <Fragment>
12       <PlanTopBar />
13       <TopBarContent>{props.children}</TopBarContent>
14     </Fragment>
15   );
16 }
```

Arquivo content.tsx

```
1  import React, { Fragment, lazy } from "react";
2  import { useSelector } from "react-redux";
3  import { getPlanView } from "../store/ui";
4
5  const SearchDisciplines = lazy(() =>
6    import("./views/search/search-disciplines")
7  );
8
9  const SearchTeams = lazy(() => import("./views/search/search-teams"));
10
11 const SearchDisciplineOffers = lazy(() =>
```

```
12    import("./views/search/search-discipline-offers")
13  );
14
15  const ManagePossibility = lazy(() =>
    ↪  import("./views/possibilities/manage"));
16
17  const ListPossibilities = lazy(() =>
    ↪  import("./views/possibilities/list"));
18
19  const Disciplines = lazy(() => import("./views/lists/list-disciplines"));
20
21  const Versions = lazy(() => import("./views/versions"));
22
23  const Calendar = lazy(() => import("./views/calendar"));
24
25  const Team = lazy(() => import("./views/details/team"));
26
27  const DisciplineOffer = lazy(() =>
    ↪  import("./views/details/discipline-offer"));
28
29  const Discipline = lazy(() => import("./views/details/discipline"));
30
31  const DisciplineOffers = lazy(() =>
32    import("./views/lists/list-discipline-offers")
33  );
34
35  const Teams = lazy(() => import("./views/lists/list-teams"));
36
37  const ManageDisciplineOffer = lazy(() =>
38    import("./views/custom/manage-discipline-offer")
39  );
40
41  const ManageTeam = lazy(() => import("./views/custom/manage-team"));
42
43  const Combinations = lazy(() => import("./views/combinations"));
44
45  const PlanContainer = lazy(() => import("./container"));
46
```

```
47 const SearchTeachers = lazy(() =>
   ↪ import("./views/search/search-teachers"));
48
49 const Teacher = lazy(() => import("./views/details/teacher"));
50
51 const Settings = lazy(() => import("./views/settings"));
52
53 export default function Content() {
54   let view = useSelector(getPlanView);
55   switch (view) {
56     case "add-possibility":
57       return <ManagePossibility key={view} edit={false} />;
58     case "edit-possibility":
59       return <ManagePossibility key={view} edit={true} />;
60     case "possibilities":
61       return <ListPossibilities key={view} />;
62     case "combinations":
63       return <Combinations key={view} />;
64     case "search-disciplines":
65       return <SearchDisciplines key={view} />;
66     case "search-teams":
67       return <SearchTeams key={view} />;
68     case "search-discipline-offers":
69       return <SearchDisciplineOffers key={view} />;
70     case "search-teachers":
71       return <SearchTeachers key={view} />;
72     case "add-discipline-offer":
73       return <ManageDisciplineOffer key={view} edit={false} />;
74     case "edit-discipline-offer":
75       return <ManageDisciplineOffer key={view} edit={true} />;
76     case "add-team":
77       return <ManageTeam key={view} edit={false} />;
78     case "settings":
79       return <Settings key={view} />;
80     case "edit-team":
81       return <ManageTeam key={view} edit={true} />;
82     case "calendar-day":
83       return <Calendar key={view} visibleDays={1} />;
84     case "calendar-three-days":
```

```
85        return <Calendar key={view} visibleDays={3} />;
86      case "calendar-week":
87        return <Calendar key={view} visibleDays={7} />;
88      case "list-disciplines":
89        return <Disciplines key={view} />;
90      case "list-discipline-offers":
91        return <DisciplineOffers key={view} />;
92      case "list-teams":
93        return <Teams key={view} />;
94      case "versions":
95        return <Versions key={view} />;
96      case "team":
97        return <Team key={view} />;
98      case "teacher":
99        return <Teacher key={view} />;
100     case "discipline-offer":
101       return <DisciplineOffer key={view} />;
102     case "discipline":
103       return <Discipline key={view} />;
104     case "404":
105     default:
106       return (
107         <PlanContainer>
108           <p>Página não encontrada</p>
109         </PlanContainer>
110       );
111   }
112 }
```

Arquivo drawer.tsx

```
1  import {
2    List,
3    ListItemText,
4    ListDivider,
5    ListItemGraphic,
6    ListGroupSubheader
7  } from "@rmwc/list";
8  import AppDrawer, {
```

```
9     Item,
10    ItemProps
11  } from "../../base/components/partials/drawer";
12  import React, { useCallback } from "react";
13  import { View, getPlanView, updateView, useUILink } from "../store/ui";
14  import { useSelector, useDispatch } from "react-redux";
15  import Possibilities from "./possibilities";
16
17  interface PlanItemProps extends Omit<ItemProps, "onClick"> {
18    id: View;
19  }
20
21  export function PlanItem(props: PlanItemProps) {
22    let onClick = useUILink(props.id);
23    return <Item onClick={onClick} {...props} />;
24  }
25
26  export default function PlanDrawer() {
27    let selectedView = useSelector(getPlanView);
28
29    return (
30      <AppDrawer selected={selectedView}>
31        <ListGroupSubheader tag="h2">Visualizações</ListGroupSubheader>
32        <List tag="nav">
33          <PlanItem id="calendar-week" selected={selectedView}>
34            <ListItemGraphic icon="today" />
35            <ListItemText>Calendário</ListItemText>
36          </PlanItem>
37          <PlanItem id="list-teams" selected={selectedView}>
38            <ListItemGraphic icon="list" />
39            <ListItemText>Lista</ListItemText>
40          </PlanItem>
41          <PlanItem id="combinations" selected={selectedView}>
42            <ListItemGraphic icon="dashboard" />
43            <ListItemText>Combinações</ListItemText>
44          </PlanItem>
45          <ListDivider />
46        </List>
47        <ListGroupSubheader tag="h2">Possibilidades</ListGroupSubheader>
```

```
48        <Possibilities />
49        <ListGroupSubheader tag="h2">Plano</ListGroupSubheader>
50        <List tag="nav">
51          <PlanItem id="versions" selected={selectedView}>
52            <ListItemGraphic icon="history" />
53            <ListItemText>Versões</ListItemText>
54          </PlanItem>
55          <PlanItem id="possibilities" selected={selectedView}>
56            <ListItemGraphic icon="collections_bookmark" />
57            <ListItemText>Possibilidades</ListItemText>
58          </PlanItem>
59          <PlanItem id="settings" selected={selectedView}>
60            <ListItemGraphic icon="settings" />
61            <ListItemText>Configurações</ListItemText>
62          </PlanItem>
63          <ListDivider />
64        </List>
65      </AppDrawer>
66    );
67  }
```

Arquivo index.tsx

```
1  import React, { Fragment, ReactNode } from "react";
2  import PlanDrawer from "./drawer";
3  import Content from "./content";
4  import { useSelector } from "react-redux";
5  import { getLoadStatus, LoadStatus } from "../store/plan";
6  import PlanContainer from "./container";
7  import { LinearProgress } from "@rmwc/linear-progress";
8
9  export default function PlanDetails() {
10   let status = useSelector(getLoadStatus);
11   let content: ReactNode;
12   switch (status) {
13     case LoadStatus.LOADING:
14       content = (
15         <PlanContainer>
16           <LinearProgress />
```

```
            <p>Carregando...</p>
          </PlanContainer>
        );
        break;
      case LoadStatus.ERROR:
        content = (
          <PlanContainer>
            <p>Erro ao carregar o plano</p>
          </PlanContainer>
        );
        break;
      case LoadStatus.LOADED:
        content = <Content />;
    }
    return (
      <Fragment>
        <PlanDrawer />
        {content}
      </Fragment>
    );
}
```

Arquivo link.tsx

```
import { View, updateView } from "../store/ui";
import { useDispatch } from "react-redux";

function useLink(view: View, idName: string) {
  let dispatch = useDispatch();
  return function(id: string) {
    return function() {
      dispatch(
        updateView({
          view,
          [idName]: id
        })
      );
    };
  };
```

```
}

export function useDisciplineLink() {
  return useLink("discipline", "discipline_id");
}

export function useTeacherLink() {
  return useLink("teacher", "teacher_id");
}

export function useDisciplineOfferLink() {
  return useLink("discipline-offer", "offer_id");
}

export function useTeamLink() {
  let dispatch = useDispatch();
  return function(offerId: string, teamId: string) {
    return function() {
      dispatch(
        updateView({
          view: "team",
          offer_id: offerId,
          team_id: teamId
        })
      );
    };
  };
}
```

Arquivo possibilities.tsx

```
import React from "react";
import {
  List,
  ListItemGraphic,
  ListItemText,
  ListDivider,
  ListItem
} from "@rmwc/list";
import { useSelector, useDispatch } from "react-redux";
```

```
10 import {
11   getSelectedPossibilityIndex,
12   getPossibilities
13 } from "../store/plan/selectors";
14 import { getQueryString } from "../../base/store/querystring";
15 import { getPlanView, updateView } from "../store/ui";
16
17 export default function Possibilities() {
18   let selectedPossibility = useSelector(getSelectedPossibilityIndex);
19   let possibilities = useSelector(getPossibilities);
20   let querystring = useSelector(getQueryString);
21   let view = useSelector(getPlanView);
22   let dispatch = useDispatch();
23   return (
24     <List tag="nav">
25       {possibilities.map((possibility, index) => {
26         return (
27           <ListItem
28             selected={index === selectedPossibility}
29             key={index}
30             onClick={() =>
31               dispatch(
32                 updateView({ ...querystring, view, possibility: "" +
                    ↪  index })
33               )
34             }
35           >
36             <ListItemGraphic icon="bookmark" />
37             <ListItemText>{possibility.name}</ListItemText>
38           </ListItem>
39         );
40       })}
41       <ListDivider />
42     </List>
43   );
44 }
```

Arquivo search-filter.tsx

```
import React, { useCallback } from "react";
import { useSelector, useDispatch } from "react-redux";
import {
  getSearchFilter,
  toggleCourses,
  togglePrerequisites
} from "../store/ui";
import { Switch } from "@rmwc/switch";
import { RemoteSearchFilter } from "../../../graphql";

export default function SearchFilter() {
  let value = useSelector(getSearchFilter);
  let dispatch = useDispatch();
  let courses = useCallback(
    function() {
      dispatch(toggleCourses());
    },
    [dispatch]
  );
  let prerequisites = useCallback(
    function() {
      dispatch(togglePrerequisites());
    },
    [dispatch]
  );
  return (
    <div style={{ padding: "1rem", textAlign: "right" }}>
      <Switch
        checked={
          value === RemoteSearchFilter.Prerequisites ||
          value === RemoteSearchFilter.All
        }
        onChange={prerequisites}
        label="Considerar pré-requisitos na filtragem?"
      />
      <Switch
        checked={
          value === RemoteSearchFilter.Courses ||
          value === RemoteSearchFilter.All
```

```
40          }
41          onChange={courses}
42          label="Mostrar apenas disciplinas dos cursos cadastrados no
            ↪  perfil"
43        />
44      </div>
45    );
46  }
```

Arquivo search.tsx

```
1   import React, { Fragment } from "react";
2   import { GridCell } from "@rmwc/grid";
3   import { LinearProgress } from "@rmwc/linear-progress";
4
5   function getCountLabel(count: number): string {
6     switch (count) {
7       case 0:
8         return "Nenhum resultado encontrado";
9       case 1:
10        return "1 resultado encontrado";
11      default:
12        return `${count} resultados encontrados`;
13    }
14  }
15
16  type Props = {
17    loading: boolean;
18    itemCount: number;
19    searchString: string;
20  };
21
22  export default function SearchString({ loading, ...props }: Props) {
23    return (
24      <Fragment>
25        <GridCell span={12}>
26          <LinearProgress closed={!loading} />
27        </GridCell>
28        <GridCell span={12}>
```

```
29          {props.searchString.length > 0
30            ? (!loading ? "Resultados para " : "Procurando por ") +
31              '"' +
32              props.searchString +
33              '"' +
34              (loading ? "" : " (" + getCountLabel(props.itemCount) + ")")
35            : loading
36            ? "Carregando..."
37            : getCountLabel(props.itemCount) + ":"}
38        </GridCell>
39      </Fragment>
40    );
41  }
```

Arquivo top-bar.tsx

```
1  import React, {
2    FormEvent,
3    useCallback,
4    useState,
5    ChangeEvent,
6    useRef,
7    useLayoutEffect,
8    ReactNode
9  } from "react";
10 import { TextField } from "@rmwc/textfield";
11 import { TopAppBarSection, TopAppBarActionItem } from
   ↪  "@rmwc/top-app-bar";
12 import TopBar from "../../base/components/partials/top-bar";
13 import {
14   getPlanView,
15   getSearchString,
16   getPlanViewTitle
17 } from "../store/ui/selectors";
18 import { useSelector, useDispatch } from "react-redux";
19 import Title from "../../base/components/title";
20 import { search, backView, updateView } from "../store/ui";
21 import { getPlanTitle } from "../store/plan/selectors";
22
```

```
type Props = {
  children?: ReactNode;
};

function prevent(e: FormEvent<Element>) {
  e.preventDefault();
}

export default function PlanTopBar(props: Props) {
  let subTitle = useSelector(getPlanViewTitle);
  let view = useSelector(getPlanView);
  let searchString = useSelector(getSearchString);
  let dispatch = useDispatch();
  let closeSearch = useCallback(() => {
    dispatch(search(""));
    dispatch(backView());
  }, [dispatch]);
  let openSearch = useCallback(() => {
    dispatch(updateView({ view: "search-teams" }));
  }, [dispatch]);

  let showSearch = view.startsWith("search-");
  let input = useRef<HTMLInputElement | null>(null);
  const formClass = "top-app-search top-app " + (!showSearch ? "hide" :
  ↪  "");
  let planName = useSelector(getPlanTitle);
  const title = planName + (subTitle ? " - " + subTitle : "");

  let notifyOnSearch = useCallback(
    (e: ChangeEvent<HTMLInputElement>) => {
      const { value } = e.target;
      dispatch(search(value));
    },
    [dispatch]
  );

  useLayoutEffect(() => {
    if (!showSearch || !input.current) {
      return;
```

```
61      }
62      input.current.focus();
63    }, [showSearch, input.current]);
64
65    return (
66      <Title title={title}>
67        <TopBar title={title} show={!showSearch}>
68          <TopAppBarSection alignEnd>
69            {props.children ?? (
70              <TopAppBarActionItem icon="search" onClick={openSearch} />
71            )}
72          </TopAppBarSection>
73        </TopBar>
74        <form onSubmit={prevent} className={formClass}>
75          <TextField
76            inputRef={input}
77            fullwidth
78            placeholder="Buscar"
79            type="text"
80            trailingIcon={{
81              tabIndex: 0,
82              icon: "close",
83              role: "button",
84              onClick: closeSearch
85            }}
86            onChange={notifyOnSearch}
87            value={searchString}
88          />
89        </form>
90      </Title>
91    );
92  }
```

B.2.54 Pasta src/js/pages/plan-details/queries

Arquivo load-plan.gql

```
1  #import "../../base/queries/table.gql"
2
3  fragment getPlanVersionData on PlanVersion {
```

```
  id
  savedAt
  selectedPossibility
  disciplines {
    deleted
    updated {
      id
      version
    }
    discipline {
      id
      version
      course {
        id
        name
      }
      habilitation {
        id
        name
      }
      step {
        id
        name
      }
      related {
        name
        type
        items {
          type
          value
          description
        }
      }
      relatedDisciplines {
        id
        course {
          id
          name
        }
```

```
        habilitation {
          id
          name
        }
        step {
          id
          name
        }
        version
        genericID
        code
        name
        type
        description
      }
      tables {
        ...table
      }
      genericID
      code
      name
      description
      type
    }
  }
  disciplineOffers {
    custom
    deleted
    updated {
      id
      version
    }
    offer {
      id
      genericID
      code
      name
      campus {
        id
```

```
      name
    }
    version
    teams {
      id
      code
      version
      schedules {
        dayOfWeek
        start {
          hour
          minute
        }
        end {
          hour
          minute
        }
        room {
          id
          genericID
          name
          description
          olcode
        }
      }
      tables {
        ...table
      }
      teachers {
        id
        genericID
        name
      }
      vacancies {
        offered
        filled
      }
      course {
        id
```

```
            name
            exclusivity
          }
        }
      }
    }
    possibilities {
      name
      disciplines {
        disciplineID
        disciplineVersion
        enabled
      }
      teams {
        offerID
        offerVersion
        teamID
        color
        selected
        enabled
      }
    }
}

fragment getPlanData on Plan {
    id
    name
    lastModified
    createdAt
    user {
      id
      name
      email
      provider: oauth2_provider
    }
    public
    university {
      id
      acronym
```

```
    name
  }
  period {
    id
    name
  }
}

query loadLatestPlanVersionID($planID: ID!) {
  plan(id: $planID) {
    id
    version: versionByIndex(index: 0) {
      id
    }
  }
}

query loadPlanVersion($planID: ID!, $planVersion: ID!) {
  plan(id: $planID) {
    ...getPlanData
    loadedVersion: versionByID(id: $planVersion) {
      ...getPlanVersionData
    }
  }
}

query loadPlanVersions($planID: ID!) {
  plan(id: $planID) {
    id
    versions {
      id
      savedAt
      totalPossibilities
      selectedSchedules {
        offerID
        color
        title
        dayOfWeek
        start {
```

```
            hour
            minute
          }
          end {
            hour
            minute
          }
        }
      }
    }
}
```

Arquivo save-plan.gql

```
#import "./load-plan.gql"
mutation savePlan($plan: PlanInput!) {
  setPlan(plan: $plan) {
    version: versionByIndex(index: 0) {
      ...getPlanVersionData
    }
  }
}
```

Arquivo search-load.gql

```
#import "../../base/queries/table.gql"

query loadDiscipline($disciplineID: ID!) {
  discipline(id: $disciplineID) {
    id
    version
    course {
      id
      name
    }
    habilitation {
      id
      name
    }
```

```
    step {
      id
      name
    }
    related {
      name
      type
      items {
        type
        value
        description
      }
    }
    relatedDisciplines {
      id
      course {
        id
        name
      }
      habilitation {
        id
        name
      }
      step {
        id
        name
      }
      genericID
      code
      version
      name
      type
      description
    }
    tables {
      ...table
    }
    genericID
    code
```

```
    name
    description
    type
  }
}

query loadDisciplineOffer($offerID: ID!) {
  disciplineOffer(id: $offerID) {
    id
    genericID
    code
    name
    campus {
      id
      name
    }
    version
    teams {
      id
      code
      version
      schedules {
        dayOfWeek
        start {
          hour
          minute
        }
        end {
          hour
          minute
        }
        room {
          id
          genericID
          name
          description
          olcode
        }
      }
```

```
      tables {
        ...table
      }
      teachers {
        id
        genericID
        name
      }
      vacancies {
        offered
        filled
      }
      course {
        id
        name
        exclusivity
      }
    }
  }
}
```

Arquivo search.gql

```
query searchTeams(
  $periodID: ID!
  $q: String!
  $offset: Int
  $version: String
  $filter: SearchFilter = NONE
) {
  searchTeams(
    periodID: $periodID
    query: { type: Terminal, value: $q }
    options: { after: $offset, version: $version, filter: $filter }
  ) {
    info {
      query {
        count
        version
      }
```

```
      page {
        after
      }
    }
    results {
      id
      code
      version
      teachers {
        id
        name
      }
      campus {
        id
        name
      }
      discipline {
        id
        code
        name
      }
      vacancies {
        offered
        filled
      }
      schedules {
        start {
          hour
          minute
        }
        end {
          hour
          minute
        }
        dayOfWeek
      }
    }
  }
}
```

```
57
58 query searchDisciplineOffers(
59   $periodID: ID!
60   $q: String!
61   $offset: Int
62   $version: String
63   $filter: SearchFilter = NONE
64 ) {
65   searchDisciplineOffers(
66     periodID: $periodID
67     query: { type: Terminal, value: $q }
68     options: { after: $offset, version: $version, filter: $filter }
69   ) {
70     info {
71       query {
72         count
73         version
74       }
75       page {
76         after
77       }
78     }
79     results {
80       id
81       code
82       name
83       campus {
84         id
85         name
86       }
87       teams {
88         id
89         code
90         vacancies {
91           offered
92           filled
93         }
94         schedules {
95           start {
```

```
            hour
            minute
          }
          end {
            hour
            minute
          }
          dayOfWeek
        }
      }
    }
  }
}

query searchDisciplines(
  $universityID: ID!
  $q: String!
  $offset: Int
  $version: String
  $filter: SearchFilter = NONE
) {
  searchDisciplines(
    universityID: $universityID
    query: { type: Terminal, value: $q }
    options: { after: $offset, version: $version, filter: $filter }
  ) {
    info {
      query {
        count
        version
      }
      page {
        after
      }
    }
    results {
      id
      code
      name
```

```
      type
      description
      course {
        id
        name
      }
    }
  }
}

query searchTeachers(
  $periodID: ID!
  $q: String!
  $offset: Int
  $version: String
) {
  searchTeachers(
    periodID: $periodID
    query: { type: Terminal, value: $q }
    options: { after: $offset, version: $version }
  ) {
    info {
      query {
        count
        version
      }
      page {
        after
      }
    }
    results {
      id
      name
      url
      teams: searchTeams(options: { limit: 1000 }) {
        results {
          id
          code
          discipline {
```

```
            id
            code
          }
          schedules {
            start {
              hour
              minute
            }
            end {
              hour
              minute
            }
            dayOfWeek
          }
        }
      }
    }
  }
}

query getAllTeams(
  $periodID: ID!
  $offset: Int
  $version: String
  $filter: SearchFilter = NONE
) {
  searchTeams(
    periodID: $periodID
    query: { type: All }
    options: { after: $offset, version: $version, filter: $filter }
  ) {
    info {
      query {
        count
        version
      }
      page {
        after
      }
```

```
    }
    results {
      id
      code
      version
      teachers {
        id
        name
      }
      campus {
        id
        name
      }
      discipline {
        id
        code
        name
      }
      vacancies {
        offered
        filled
      }
      schedules {
        start {
          hour
          minute
        }
        end {
          hour
          minute
        }
        dayOfWeek
      }
    }
  }
}

query getAllTeachers($periodID: ID!, $offset: Int, $version: String) {
  searchTeachers(
```

```
    periodID: $periodID
    query: { type: All }
    options: { after: $offset, version: $version }
  ) {
    info {
      query {
        count
        version
      }
      page {
        after
      }
    }
    results {
      id
      name
      url
      teams: searchTeams(options: { limit: 1000 }) {
        results {
          id
          code
          discipline {
            id
            code
          }
          schedules {
            start {
              hour
              minute
            }
            end {
              hour
              minute
            }
            dayOfWeek
          }
        }
      }
    }
```

```
  }
}

query getAllDisciplineOffers(
  $periodID: ID!
  $offset: Int
  $version: String
  $filter: SearchFilter = NONE
) {
  searchDisciplineOffers(
    periodID: $periodID
    query: { type: All }
    options: { version: $version, after: $offset, filter: $filter }
  ) {
    info {
      query {
        count
        version
      }
      page {
        after
      }
    }
    results {
      id
      code
      name
      campus {
        id
        name
      }
      teams {
        id
        code
        vacancies {
          offered
          filled
        }
        schedules {
```

```
330           start {
331             hour
332             minute
333           }
334           end {
335             hour
336             minute
337           }
338           dayOfWeek
339         }
340       }
341     }
342   }
343 }
344
345 query getAllDisciplines(
346   $universityID: ID!
347   $offset: Int
348   $version: String
349   $filter: SearchFilter = NONE
350 ) {
351   searchDisciplines(
352     universityID: $universityID
353     query: { type: All }
354     options: { version: $version, after: $offset, filter: $filter }
355   ) {
356     info {
357       query {
358         count
359         version
360       }
361       page {
362         after
363       }
364     }
365     results {
366       id
367       code
368       name
```

```
      type
      description
      course {
        id
        name
      }
    }
  }
}
```

B.2.55 Pasta src/js/pages/plan-details/store

Arquivo index.ts

```
import { IModule } from "redux-dynamic-modules";
import getUIModule, { FullState as UIState } from "../../base/store/ui";
import getAuthModule, { FullState as AuthState } from
↪  "../../base/store/auth";
import getQueryStringModule from "../../base/store/querystring";
import { FullState as CSRFState } from "../../base/store/csrf";
import getPlanModule, { FullState as PlanState } from "./plan";
import getPlanUIModule, { FullState as PlanUIState } from "./ui";
import getPlanCombinationsModule, {
  FullState as PlanCombinationsState
} from "./combinations";

type Module =
  | IModule<
      | UIState
      | PlanState
      | AuthState
      | CSRFState
      | PlanUIState
      | PlanCombinationsState
    >
  | Module[];

export default function getReduxModule(): Module[] {
  return [
    getQueryStringModule(),
```

```
    [getAuthModule(), getPlanModule()],
    getUIModule(),
    getPlanCombinationsModule(),
    getPlanUIModule()
  ];
}
```

B.2.56 Pasta src/js/pages/plan-list/components

Arquivo index.tsx

```tsx
import React, { useCallback } from "react";
import { Grid, GridCell } from "@rmwc/grid";
import { useQuery } from "@apollo/react-hooks";
import getPlans from "../queries/get-plans.gql";
import { Typography } from "@rmwc/typography";
import { CardPrimaryAction, Card } from "@rmwc/card";
import {
  RemoteGetPlansQuery,
  RemoteGetPlansQueryVariables,
  RemotePlan,
  RemoteUniversity,
  RemotePeriod,
  RemotePlanVersion
} from "../../../graphql";
import { useDispatch } from "react-redux";
import { redirect } from "../../base/store/querystring";
import { Button } from "@rmwc/button";
import { push } from "connected-react-router";
import { PlanCard } from "../../base/components/cards/plan";
import { CalendarProvider } from "../../base/components/calendar";
import { useRedirectIfNotAuthenticated } from
↪  "../../base/components/auth";

export type PlanProps = {
  plan: Pick<RemotePlan, "id" | "name" | "createdAt" | "lastModified"> &
  ↪  {
    university: Pick<RemoteUniversity, "id" | "name">;
    period: Pick<RemotePeriod, "id" | "name">;
  };
```

```
28    latestVersion?: Pick<
29      RemotePlanVersion,
30      "totalPossibilities" | "selectedSchedules"
31    >;
32  };
33
34  export function Plan({ plan, latestVersion }: PlanProps) {
35    let dispatch = useDispatch();
36    let onDetails = useCallback(
37      function() {
38        dispatch(
39          redirect({
40            url: "/planos/editar",
41            querystring: {
42              id: plan.id
43            }
44          })
45        );
46      },
47      [dispatch, plan]
48    );
49    return (
50      <GridCell desktop={4} phone={4} tablet={4}>
51        <PlanCard plan={plan} onDetails={onDetails} version={latestVersion}
          ↪ />
52      </GridCell>
53    );
54  }
55  export default function PlanList() {
56    useRedirectIfNotAuthenticated();
57    let { loading, error, data } = useQuery<
58      RemoteGetPlansQuery,
59      RemoteGetPlansQueryVariables
60    >(getPlans);
61    let dispatch = useDispatch();
62    if (loading) {
63      return <p>Carregando dados...</p>;
64    }
65    if (error || !data) {
```

```
66      return <p>Houve um erro durante o carregamento dos dados</p>;
67    }
68    return (
69      <Grid>
70        <CalendarProvider width={344} height={194}>
71          {data.plans.map(plan => (
72            <Plan plan={plan} key={plan.id}
              ↪  latestVersion={plan.latestVersion} />
73          ))}
74          {data.plans.length === 0 ? (
75            <GridCell span={12}>
76              Parece que você não tem planos cadastrados.
77              <br />
78              <br />
79              <Button onClick={() => dispatch(push("/planos/cadastrar/"))}>
80                Deseja cadastrar um novo plano?
81              </Button>
82            </GridCell>
83          ) : null}
84        </CalendarProvider>
85      </Grid>
86    );
87  }
```

B.2.57 Pasta src/js/pages/plan-list/queries

Arquivo get-plans.gql

```
1  query getPlans {
2    plans {
3      id
4      name
5      createdAt
6      lastModified
7      university {
8        id
9        name
10     }
11     period {
12       id
```

```
      name
    }
    latestVersion: versionByIndex(index: 0) {
      id
      selectedSchedules {
        offerID
        title
        color
        dayOfWeek
        start {
          hour
          minute
        }
        end {
          hour
          minute
        }
      }
      totalPossibilities
    }
  }
}
```

B.2.58 Pasta src/js/pages/signup/components

Arquivo index.tsx

```
import React from "react";
import { useTextField, useSubmit } from "../../base/components/form";
import { Button } from "@rmwc/button";
import { BASE_BACKEND_URL } from "../../../config";
import { GridCell, Grid, GridInner } from "@rmwc/grid";
import { Link } from "react-router-dom";
import { TextField } from "@rmwc/textfield";
import { Snackbar } from "@rmwc/snackbar";
import { useRedirectIfAuthenticated } from "../../base/components/auth";

interface Fields {
  name: string;
  email: string;
```

```
14    password: string;
15    confirm_password: string;
16  }
17
18  async function submit(
19    csrfToken: string | null,
20    fields: Fields
21  ): Promise<boolean> {
22    let error: string | null =
23      "Erro durante o cadastro. Tente novamente, por favor.";
24    try {
25      if (fields.password.length < 8) {
26        error = "A senha deve ter no mínimo 8 caracteres";
27      } else if (!/(?=.*[A-Z])/.test(fields.password)) {
28        error = "A senha deve ter pelo menos um caractere maiusculo";
29      } else if (!/(?=.*[a-z])/.test(fields.password)) {
30        error = "A senha deve ter pelo menos um caractere minusculo";
31      } else if (!/(?=.*[0-9])/.test(fields.password)) {
32        error = "A senha deve ter pelo menos um digito numérico";
33      } else if (!/(?=.[!@#\$%\^&])/.test(fields.password)) {
34        error = "A senha deve ter pelo menos um caractere especial";
35      } else if (fields.password !== fields.confirm_password) {
36        error = "Senha e confirmação de senha precisam ser iguais";
37      } else if (!csrfToken) {
38        error = "Erro interno ao enviar requisição. Tente recarregar a
          ↪ página.";
39      } else {
40        let response: Response = await fetch(
41          `${BASE_BACKEND_URL}/auth/register`,
42          {
43            body: JSON.stringify(fields),
44            method: "POST",
45            credentials: "include",
46            headers: {
47              "X-CSRF-Token": csrfToken
48            }
49          }
50        );
51        if (response.ok) {
```

```
52          interface APIResponse {
53            status: "success" | "failure";
54            error?: string;
55          }
56          var data: APIResponse = await response.json();
57          if (data.status === "success") {
58            error = null;
59          }
60        }
61      }
62    } catch (e) {
63      error = "Erro desconhecido durante o cadastro.";
64    }
65    return error === null || Promise.reject(error);
66  }
67
68  export default function SignupHook() {
69    let [name, nameField] = useTextField();
70    let [email, emailField] = useTextField();
71    let [password, passwordField] = useTextField();
72    let [confirm_password, confirmPasswordField] = useTextField();
73    useRedirectIfAuthenticated();
74    let { submitted, error, clearError, onSubmit } = useSubmit((evt,
      ↪  csrfToken) =>
75      submit(csrfToken, { name, email, password, confirm_password })
76    );
77    if (submitted) {
78      return (
79        <p>
80          Cadastro feito com sucesso. Verifique o seu e-mail para confirmar
            ↪  sua
81          conta.
82        </p>
83      );
84    }
85    return (
86      <form onSubmit={onSubmit}>
87        <Grid>
88          <GridCell span={12} className="form-row">
```

```
89            Use o formulário abaixo para criar sua conta no Guru da
              ↪ Matrícula. Ou
90            se autentique usando sua página do Facebook ou Google
91            <a href="/entrar">na página de autenticação</a>.
92          </GridCell>
93          <GridCell span={12} className="form-row">
94            <TextField
95              name="name"
96              type="text"
97              label="Nome"
98              autoComplete="name"
99              icon="person"
100             required
101             {...nameField}
102           />
103         </GridCell>
104         <GridCell span={12} className="form-row">
105           <TextField
106             name="email"
107             type="email"
108             label="E-mail"
109             autoComplete="email"
110             icon="email"
111             required
112             {...emailField}
113           />
114         </GridCell>
115         <GridCell span={12} className="form-row">
116           <TextField
117             name="password"
118             type="password"
119             label="Senha"
120             autoComplete="new-password"
121             icon="lock"
122             minLength={8}
123             required
124             {...passwordField}
125           />
126         </GridCell>
```

```
127           <GridCell span={12} className="form-row">
128             <GridInner>
129               <GridCell span={4} align="middle">
130                 <TextField
131                   name="confirm_password"
132                   type="password"
133                   label="Confirmar Senha"
134                   autoComplete="new-password"
135                   icon="lock"
136                   required
137                   {...confirmPasswordField}
138                 />
139               </GridCell>
140             </GridInner>
141           </GridCell>
142           <GridCell span={12} className="form-row">
143             <Button type="submit" raised>
144               Cadastrar
145             </Button>
146           </GridCell>
147           <GridCell span={12} className="form-row">
148             Lembrou seu e-mail e senha?
149             <Link to="/entrar">Entre com sua conta do Guru da
                ↪  Matrícula.</Link>
150           </GridCell>
151           <GridCell span={12} className="form-row">
152             Tem conta, mas se esqueceu da senha?
153             <Link to="/recuperar-senha">Recupere a sua conta.</Link>
154           </GridCell>
155         </Grid>
156         {error ? <Snackbar open onClose={clearError} message={error} /> :
            ↪  null}
157       </form>
158     );
159   }
```

B.2.59 Pasta src/js/pages/teacher/components

Arquivo index.tsx

```
import React, { Fragment } from "react";
import { useSelector } from "react-redux";
import { getQueryString } from "../../base/store/querystring";
import { Grid, GridCell } from "@rmwc/grid";
import { Typography } from "@rmwc/typography";
import { useQuery } from "@apollo/react-hooks";
import {
  RemoteGetTeacherQuery,
  RemoteGetTeacherQueryVariables
} from "../../../graphql";
import Title from "../../base/components/title";
import TeamCard from "../../base/components/cards/team";
import { CalendarProvider } from "../../base/components/calendar";
import { getTeacher } from "../queries//get-teacher.gql";

export type TeacherDetailsModel = Exclude<
  RemoteGetTeacherQuery["teacher"]["results"][0],
  undefined
>;

export type TeacherTeamDetailsModel =
  ↪ TeacherDetailsModel["teams"]["results"][0];

type TeacherDetailsProps = {
  teacher: TeacherDetailsModel;
  onDetails?: (
    teacher: TeacherDetailsModel,
    team: TeacherTeamDetailsModel
  ) => void;
  onAction?: (
    teacher: TeacherDetailsModel,
    team: TeacherTeamDetailsModel
  ) => void;
  isEnabled?: (
    teacher: TeacherDetailsModel,
    team: TeacherTeamDetailsModel
  ) => boolean;
  isSelected?: (
    teacher: TeacherDetailsModel,
```

```
39      team: TeacherTeamDetailsModel
40    ) => boolean;
41  };
42
43  export function TeacherDetails(props: TeacherDetailsProps) {
44    let { teacher, onDetails, onAction, isEnabled, isSelected } = props;
45    return (
46      <Grid className="team">
47        <GridCell span={12}>
48          <Typography use="headline3">{teacher.name}</Typography>
49        </GridCell>
50        <GridCell desktop={2} phone={4} tablet={4}>
51          <Typography use="headline5">Universidade:</Typography>
52        </GridCell>
53        <GridCell desktop={10} phone={4} tablet={4}>
54          <Typography use="body1">{teacher.university.name}</Typography>
55        </GridCell>
56        <GridCell desktop={2} phone={4} tablet={4}>
57          <Typography use="headline5">Período:</Typography>
58        </GridCell>
59        <GridCell desktop={10} phone={4} tablet={4}>
60          <Typography use="body1">{teacher.period.name}</Typography>
61        </GridCell>
62        {teacher.url ? (
63          <Fragment>
64            <GridCell desktop={2} phone={4} tablet={4}>
65              <Typography use="headline5">Site:</Typography>
66            </GridCell>
67            <GridCell desktop={10} phone={4} tablet={4}>
68              <Typography use="body1">
69                <a href={teacher.url} rel="nofollow" target="_blank">
70                  {teacher.url}
71                </a>
72              </Typography>
73            </GridCell>
74          </Fragment>
75        ) : null}
76        <GridCell span={12}>
```

```
77          <Typography use="headline5">Turmas:</Typography>
78        </GridCell>
79        <CalendarProvider width={344} height={194}>
80          {teacher.teams.results.map(team => (
81            <GridCell span={4} key={team.id}>
82              <TeamCard
83                onDetails={
84                  onDetails
85                    ? () => onDetails && onDetails(teacher, team)
86                    : undefined
87                }
88                onAction={
89                  onAction ? () => onAction && onAction(teacher, team) :
                    ↪ undefined
90                }
91                enabled={isEnabled && isEnabled(teacher, team)}
92                selected={isSelected && isSelected(teacher, team)}
93                team={team}
94              />
95            </GridCell>
96          ))}
97        </CalendarProvider>
98      </Grid>
99    );
100 }
101
102 type RemoteTeacherDetailsProps = Omit<TeacherDetailsProps, "teacher"> & {
103   periodID: string;
104   teacherID: string;
105   changeTitle?: boolean;
106 };
107
108 export function RemoteTeacherDetails(props: RemoteTeacherDetailsProps) {
109   let { changeTitle, periodID, teacherID, ...events } = props;
110   let { loading, data, error } = useQuery<
111     RemoteGetTeacherQuery,
112     RemoteGetTeacherQueryVariables
113   >(getTeacher, {
114     variables: {
```

```
115       periodID,
116       teacherID
117     }
118   });
119   if (error) {
120     return <p>Houve um erro durante o carregamento dos dados.</p>;
121   }
122   if (loading) {
123     return <p>Carregando professor...</p>;
124   }
125   if (!data || !data.teacher || !data.teacher.results.length) {
126     return <p>Professor não encontrado</p>;
127   }
128   let teacher = data.teacher.results[0];
129   let result = <TeacherDetails teacher={teacher} {...events} />;
130   if (!changeTitle) {
131     return result;
132   }
133   return <Title title={teacher.name}>{result}</Title>;
134 }
135
136 export default function TeacherPage() {
137   let queryString = useSelector(getQueryString);
138   let periodId = queryString["period_id"];
139   let id = queryString["id"];
140   if (typeof id !== "string" || typeof periodId !== "string") {
141     return <p>Requisição inválida.</p>;
142   }
143   return (
144     <RemoteTeacherDetails periodID={periodId} teacherID={id} changeTitle
      ↪ />
145   );
146 }
```

B.2.60 Pasta src/js/pages/teacher/queries

Arquivo get-teacher.gql

```
1 query getTeacher($periodID: ID!, $teacherID: String!) {
2   teacher: searchTeachers(
```

```
    periodID: $periodID
    query: { type: FieldTerminal, attribute: "id", value: $teacherID }
    options: { limit: 1 }
  ) {
    results {
      id
      university {
        id
        name
      }
      period {
        id
        name
      }
      name
      url
      teams: searchTeams(options: { limit: 1000 }) {
        results {
          id
          code
          discipline {
            id
            code
            name
          }
          campus {
            id
            name
          }
          schedules {
            start {
              hour
              minute
            }
            end {
              hour
              minute
            }
            dayOfWeek
```

```
42            }
43          }
44        }
45      }
46    }
47  }
```

B.2.61 Pasta src/js/pages/team/components

Arquivo index.tsx

```
1  import React, {
2    useEffect,
3    Fragment,
4    useState,
5    ReactNode,
6    useCallback
7  } from "react";
8  import { useSelector, useDispatch } from "react-redux";
9  import { getQueryString, redirect } from "../../base/store/querystring";
10 import { Grid, GridCell } from "@rmwc/grid";
11 import { Typography } from "@rmwc/typography";
12 import { useQuery } from "@apollo/react-hooks";
13 import getTeam from "../queries/get-team.gql";
14 import {
15   RemoteGetTeamQuery,
16   RemoteGetTeamQueryVariables,
17   RemoteExclusivity
18 } from "../../../graphql";
19 import Title from "../../base/components/title";
20 import { Table } from "../../base/components/table";
21 import { CalendarInteractive, Schedule } from
   ↪  "../../base/components/calendar";
22 import { List, ListItem, ListItemText } from "@rmwc/list";
23 import { SimpleDialog } from "@rmwc/dialog";
24
25 type InternalTeam = Exclude<RemoteGetTeamQuery["team"], undefined>;
26 export type TeamDetailsModel = Omit<
27   InternalTeam,
28   "schedules" | "campus" | "discipline"
```

```
> & {
  discipline: Omit<InternalTeam["discipline"], "genericID">;
  campus?: InternalTeam["campus"];
  schedules: (Omit<InternalTeam["schedules"][0], "room"> & {
    room?: InternalTeam["schedules"][0]["room"];
  })[];
};

type TeamDetailsProps = {
  team: TeamDetailsModel;
  useTeacherLink?: (teacher: TeamDetailsModel["teachers"][0]) => () =>
  ↪ void;
  useDiscipline?: (disciplnie: TeamDetailsModel["discipline"]) => () =>
  ↪ void;
};

function TeamTables({ team }: TeamDetailsProps) {
  let { tables } = team;
  if (!tables || !tables.length) {
    return null;
  }
  return (
    <Fragment>
      <GridCell span={12}>
        <Typography use="headline5">Outras informações:</Typography>
      </GridCell>
      {tables.map(table => (
        <GridCell span={12} key={table.id}>
          <Table table={table} />
        </GridCell>
      ))}
    </Fragment>
  );
}

function TeamCourse({ team }: TeamDetailsProps) {
  let { course } = team;
  if (!course) {
    return null;
```

```
66    }
67    let exclusivity: string;
68    switch (course.exclusivity) {
69      case RemoteExclusivity.Exclusive:
70        exclusivity = "Exclusiva";
71        break;
72      case RemoteExclusivity.NotExclusive:
73        exclusivity = "Não-exclusiva";
74        break;
75      default:
76        exclusivity = "Desconhecida";
77    }
78    return (
79      <Fragment>
80        <GridCell desktop={2} phone={4} tablet={4}>
81          <Typography use="headline5">Curso:</Typography>
82        </GridCell>
83        <GridCell desktop={10} phone={4} tablet={4}>
84          <Typography use="body1">
85            {course.name} (exclusividade: {exclusivity})
86          </Typography>
87        </GridCell>
88      </Fragment>
89    );
90  }
91
92  function TeamVacancies({ team }: TeamDetailsProps) {
93    let { vacancies } = team;
94    if (!vacancies) {
95      return null;
96    }
97    return (
98      <Fragment>
99        <GridCell desktop={2} phone={4} tablet={4}>
100         <Typography use="headline5">Vagas:</Typography>
101       </GridCell>
102       <GridCell desktop={10} phone={4} tablet={4}>
103         <Typography use="body1">
```

```
104             {vacancies.filled} vagas preenchidas/{vacancies.offered} vagas
                ↪  totais
105           </Typography>
106         </GridCell>
107       </Fragment>
108     );
109   }
110
111   type TeamSchedulesProps = {
112     schedules: TeamDetailsModel["schedules"];
113     roomId: string | null;
114     onClose: () => void;
115   };
116
117   function TeamRoom(props: TeamSchedulesProps) {
118     let actualRoom: TeamDetailsModel["schedules"][0]["room"];
119     let body: ReactNode;
120     for (let { room } of props.schedules) {
121       if (props.roomId !== null && room && room.id === props.roomId) {
122         actualRoom = room;
123         body = (
124           <Fragment>
125             <b>Nome: </b> {actualRoom.name}
126             <br />
127             <b>Descrição: </b> {actualRoom.description}
128             <br />
129             <b>OLCode: </b> {actualRoom.olcode}
130           </Fragment>
131         );
132         break;
133       }
134     }
135     return (
136       <SimpleDialog
137         title={actualRoom?.name}
138         body={body}
139         open={!!actualRoom}
140         onClose={props.onClose}
141         acceptLabel={null}
```

```
142       cancelLabel="Fechar"
143     />
144   );
145 }
146
147 export function TeamDetails(props: TeamDetailsProps) {
148   let { team, useTeacherLink, useDiscipline } = props;
149   let [roomId, setRoomId] = useState<string | null>(null);
150   let schedules: Schedule[] = [];
151   for (let schedule of team.schedules) {
152     schedules.push({
153       ...schedule,
154       id: schedule?.room?.id ?? "",
155       title: schedule?.room?.name ?? ""
156     });
157   }
158   let discipline = useDiscipline ? useDiscipline(team.discipline) :
    ↪  undefined;
159   return (
160     <Fragment>
161       <TeamRoom
162         roomId={roomId}
163         onClose={() => setRoomId(null)}
164         schedules={team.schedules}
165       />
166       <Grid className="team">
167         <GridCell span={12}>
168           <Typography use="headline3">{team.code}</Typography>
169         </GridCell>
170         <GridCell span={12}>
171           <Typography use="subtitle1">
172             {discipline ? (
173               <a href="#" onClick={discipline}>
174                 {team.discipline.code} - {team.discipline.name}
175               </a>
176             ) : (
177               team.discipline.code + " - " + team.discipline.name
178             )}
179           </Typography>
```

```
180           </GridCell>
181           <TeamCourse {...props} />
182           <TeamVacancies {...props} />
183           <GridCell span={12}>
184             <Typography use="headline5">Horários:</Typography>
185           </GridCell>
186           <GridCell span={12}>
187             <CalendarInteractive
188               schedules={schedules}
189               slide
190               onScheduleClick={schedule => setRoomId(schedule.id ?? null)}
191             />
192           </GridCell>
193           <GridCell desktop={2} phone={4} tablet={4}>
194             <Typography use="headline5">Universidade:</Typography>
195           </GridCell>
196           <GridCell desktop={10} phone={4} tablet={4}>
197             <Typography use="body1">{team.university.name}</Typography>
198           </GridCell>
199           <GridCell desktop={2} phone={4} tablet={4}>
200             <Typography use="headline5">Período:</Typography>
201           </GridCell>
202           <GridCell desktop={10} phone={4} tablet={4}>
203             <Typography use="body1">{team.period.name}</Typography>
204           </GridCell>
205           <GridCell desktop={2} phone={4} tablet={4}>
206             <Typography use="headline5">Professores:</Typography>
207           </GridCell>
208           <GridCell desktop={10} phone={4} tablet={4}>
209             <List>
210               {team.teachers.map(teacher => (
211                 <ListItem
212                   key={teacher.id}
213                   onClick={
214                     useTeacherLink
215                       ? () => useTeacherLink && useTeacherLink(teacher)
216                       : undefined
217                   }
218                 >
```

```
219                 <ListItemText>
220                   <Typography use="body1">{teacher.name}</Typography>
221                 </ListItemText>
222               </ListItem>
223             ))}
224           </List>
225         </GridCell>
226         <TeamTables {...props} />
227       </Grid>
228     </Fragment>
229   );
230 }
231
232 type RemoteTeamDetailsProps = {
233   offerId: string;
234   teamId: string;
235   changeTitle?: boolean;
236   useTeacherLink?: (teacher: TeamDetailsModel["teachers"][0]) => () =>
    ↪  void;
237   useDiscipline?: (discipline: TeamDetailsModel["discipline"]) => () =>
    ↪  void;
238 };
239
240 export function RemoteTeamDetails(props: RemoteTeamDetailsProps) {
241   let { changeTitle, useDiscipline, useTeacherLink, ...variables } =
    ↪  props;
242   let { loading, data, error } = useQuery<
243     RemoteGetTeamQuery,
244     RemoteGetTeamQueryVariables
245   >(getTeam, {
246     variables
247   });
248   if (error) {
249     return <p>Houve um erro durante o carregamento dos dados.</p>;
250   }
251   if (loading) {
252     return <p>Carregando turma...</p>;
253   }
254   if (!data || !data.team) {
```

```
255       return <p>Turma não encontrada</p>;
256     }
257     let { team } = data;
258     let result = (
259       <TeamDetails
260         team={team}
261         useDiscipline={useDiscipline}
262         useTeacherLink={useTeacherLink}
263       />
264     );
265     if (!changeTitle) {
266       return result;
267     }
268     return (
269       <Title
270         title={
271           team.code + " - " + team.discipline.code + " - " +
              ↪  team.discipline.name
272         }
273       >
274         {result}
275       </Title>
276     );
277   }
278
279   export default function TeamPage() {
280     let queryString = useSelector(getQueryString);
281     let offerId = queryString["offer_id"];
282     let teamId = queryString["team_id"];
283     let dispatch = useDispatch();
284     let discipline = useCallback(function() {
285       return function() {
286         dispatch(
287           redirect({
288             url: "/disciplinas/detalhes",
289             querystring: { id: offerId }
290           })
291         );
292       };
```

```
293   }, []);
294   if (typeof offerId !== "string" || typeof teamId !== "string") {
295     return (
296       <p>Requisição inválida. Parâmetros requeridos não foram
          ↪  especificados.</p>
297     );
298   }
299   return (
300     <RemoteTeamDetails
301       offerId={offerId}
302       teamId={teamId}
303       changeTitle={true}
304       useDiscipline={discipline}
305     />
306   );
307 }
```

B.2.62 Pasta src/js/pages/team/queries

Arquivo get-team.gql

```
1  #import "../../base/queries/table.gql"
2  query getTeam($offerId: ID!, $teamId: ID!) {
3    team(disciplineID: $offerId, teamID: $teamId) {
4      id
5      university {
6        id
7        name
8      }
9      period {
10       id
11       name
12     }
13     campus {
14       id
15       name
16     }
17     course {
18       id
19       name
```

```
      exclusivity
    }
    vacancies {
      offered
      filled
    }
    discipline {
      id
      code
      genericID
      name
    }
    code
    tables {
      ...table
    }
    schedules {
      start {
        hour
        minute
      }
      end {
        hour
        minute
      }
      dayOfWeek
      room {
        id
        genericID
        name
        description
        olcode
      }
    }
    teachers {
      id
      name
    }
  }
```

```
59 }
```

B.2.63 Pasta src/js/pages/universities/components

Arquivo index.tsx

```
1  import React from "react";
2  import {
3    DataTable,
4    DataTableContent,
5    DataTableHead,
6    DataTableBody,
7    DataTableRow,
8    DataTableHeadCell,
9    DataTableCell
10 } from "@rmwc/data-table";
11 import { useQuery } from "@apollo/react-hooks";
12 import universitiesQuery from "../queries/universities.gql";
13 import {
14   RemoteGetOwnedUniversitiesQuery,
15   RemoteGetOwnedUniversitiesQueryVariables
16 } from "../../../graphql";
17 import { useRedirectIfNotAuthenticated } from
   ↪  "../../base/components/auth";
18 import { Button } from "@rmwc/button";
19 import { useDispatch } from "react-redux";
20 import { push } from "connected-react-router";
21 import {
22   DesktopOnly,
23   TabletDesktop
24 } from "../../base/components/partials/responsive";
25
26 function formatDate(date?: Date): string {
27   if (!date) {
28     return "Nunca";
29   }
30   return new Date(date).toLocaleDateString("pt-br", {
31     day: "numeric",
32     weekday: "long",
33     year: "numeric",
```

```
34      month: "numeric",
35      hour: "numeric",
36      minute: "numeric"
37    });
38  }
39
40  export default function Universities() {
41    let dispatch = useDispatch();
42    useRedirectIfNotAuthenticated();
43    let { loading, data, error } = useQuery<
44      RemoteGetOwnedUniversitiesQuery,
45      RemoteGetOwnedUniversitiesQueryVariables
46    >(universitiesQuery);
47    if ((loading || !data) && !error) {
48      return <p>Carregando...</p>;
49    }
50    if (error || !data) {
51      return <p>Houve um erro durante o carregamento dos dados</p>;
52    }
53    return (
54      <DataTable>
55        <DataTableContent>
56          <DataTableHead>
57            <DataTableRow>
58              <DataTableHeadCell>Acrônimo</DataTableHeadCell>
59              <TabletDesktop>
60                <DataTableHeadCell>Nome</DataTableHeadCell>
61              </TabletDesktop>
62              <DataTableHeadCell>Pública</DataTableHeadCell>
63              <DesktopOnly>
64                <DataTableHeadCell>
65                  Última Atualização de Ofertas
66                </DataTableHeadCell>
67                <DataTableHeadCell>
68                  Última Atualização de Habilitações
69                </DataTableHeadCell>
70              </DesktopOnly>
71              <DataTableHeadCell colSpan={2}>Ações</DataTableHeadCell>
72            </DataTableRow>
```

```
73          </DataTableHead>
74          <DataTableBody>
75            {data.universities.map(row => (
76              <DataTableRow key={row.id}>
77                <DataTableCell>{row.acronym}</DataTableCell>
78                <TabletDesktop>
79                  <DataTableCell>{row.name}</DataTableCell>
80                </TabletDesktop>
81                <DataTableCell>{row.public ? "Sim" : "Não"}</DataTableCell>
82                <DesktopOnly>
83                  <DataTableCell>
84                    {formatDate(row.offers.lastUpdated)}
85                  </DataTableCell>
86                  <DataTableCell>
87                    {formatDate(row.habilitations.lastUpdated)}
88                  </DataTableCell>
89                </DesktopOnly>
90                <DataTableCell>
91                  <Button
92                    onClick={() =>
93                      dispatch(
94                        push(
95                          "/universidades/editar?id=" +
                            ↪  encodeURIComponent(row.id)
96                        )
97                      )
98                    }
99                  >
100                   Editar
101                 </Button>
102               </DataTableCell>
103             </DataTableRow>
104           ))}
105           {data.universities.length === 0 ? (
106             <DataTableRow>
107               <DataTableCell colSpan={8}>
108                 Parece que você não tem universidades cadastradas.
109                 <br />
110                 <br />
```

```
111                   <Button
112                     onClick={() =>
                        ↪  dispatch(push("/universidades/cadastrar"))}
113                   >
114                     Deseja cadastrar uma universidade?
115                   </Button>
116                 </DataTableCell>
117               </DataTableRow>
118             ) : null}
119           </DataTableBody>
120         </DataTableContent>
121       </DataTable>
122     );
123   }
```

B.2.64 Pasta src/js/pages/universities/queries

Arquivo universities.gql

```
1  query GetOwnedUniversities {
2    universities(onlyOwned: true) {
3      id
4      acronym
5      name
6      public
7      offers {
8        lastUpdated
9      }
10     habilitations {
11       lastUpdated
12     }
13   }
14 }
```

B.2.65 Pasta src/js/pages/base/components/calendar

Arquivo index.tsx

```
1  export * from "./interactive";
2  export * from "./thumbnail";
```

```
export { Schedule } from "./types";
```

Arquivo interactive.dimension.tsx

```
import { useEffect, useState, MutableRefObject } from "react";
import { Dimension } from "./types";

export function useContainerDimension(
  container: MutableRefObject<HTMLDivElement | null>
) {
  let [dimension, setDimension] = useState<Dimension | null>(null);

  useEffect(() => {
    let updateContainerDimension = function() {
      if (!container || !container.current) {
        return;
      }
      const { width, height } =
      ↪  container.current.getBoundingClientRect();
      if (
        !dimension ||
        width !== dimension.width ||
        height !== dimension.height
      ) {
        setDimension({ width, height });
      }
    };

    window.addEventListener("resize", updateContainerDimension, false);
    updateContainerDimension();
    return () => {
      window.removeEventListener("resize", updateContainerDimension,
      ↪  false);
    };
  }, [container]);
  return dimension;
}
```

Arquivo interactive.height.tsx

```tsx
import { useState, MutableRefObject, useEffect } from "react";

export function useViewportHeight(
  ref: MutableRefObject<HTMLDivElement | null>
) {
  let [height, setHeight] = useState<number>(0);

  useEffect(
    function() {
      function updateHeight() {
        if (!ref.current) {
          return;
        }
        let viewportHeight = Math.max(
          document.documentElement.clientHeight,
          window.innerHeight || 0
        );
        let refRect = ref.current.getBoundingClientRect();
        setHeight(viewportHeight - refRect.top);
      }
      window.addEventListener("resize", updateHeight, false);
      updateHeight();
      return () => {
        window.removeEventListener("resize", updateHeight, false);
      };
    },
    [ref]
  );
  return height;
}
```

Arquivo interactive.select.tsx

```tsx
import {
  CalendarSettings,
  DayHourMinute,
  DayInterval,
  HourMinute
} from "./types";
import { CalendarInteractiveProps } from "./interactive";
```

```
import {
  useCallback,
  useState,
  MouseEvent,
  TouchEvent,
  useReducer
} from "react";
import {
  isEqual,
  isBefore,
  toHourMinute,
  hasConflict,
  toMinutes
} from "./utilities";

type CalendarSelectOptions = {
  props: Pick<
    CalendarInteractiveProps,
    "onScheduleClick" | "onSelect" | "schedules"
  >;
  settings: CalendarSettings;
};

function getDayInterval(
  target: HTMLElement,
  y: number,
  { hourHeight, interval }: CalendarSettings
): DayInterval | null {
  let { dayOfWeek } = target.dataset;
  if (!dayOfWeek) {
    return null;
  }
  let rect = target.getBoundingClientRect();
  let intervalSize = (hourHeight / 60) * interval;
  let diff = y - rect.top;
  let base = diff / intervalSize;
  let end = Math.ceil(base) * interval;
  end = Math.max(end, interval);
  end = Math.min(end, 24 * 60);
```

```
  let start = Math.floor(base) * interval;
  start = Math.max(start, 0);
  start = Math.min(start, 24 * 60 - interval);
  return {
    dayOfWeek: parseInt(dayOfWeek),
    start: toHourMinute(start),
    end: toHourMinute(end)
  };
}

const enum selectionDirection {
  FORWARD,
  BACKWARD
}

type selectionState =
  | (DayInterval & {
      direction: selectionDirection;
    })
  | null;

type selectionAction =
  | {
      type: "start";
      interval: DayInterval;
    }
  | {
      type: "move";
      settings: CalendarSettings;
      interval: DayInterval;
      schedules: CalendarInteractiveProps["schedules"];
    }
  | {
      type: "clear";
    };

function selectionStateReducer(
  state: selectionState,
  action: selectionAction
```

```
86 ): selectionState {
87   switch (action.type) {
88     case "clear":
89       return null;
90     case "start":
91       return {
92         ...action.interval,
93         direction: selectionDirection.FORWARD
94       };
95     case "move":
96       if (state === null) {
97         return state;
98       }
99       if (state.dayOfWeek !== action.interval.dayOfWeek) {
100        return state;
101      }
102      let startBefore = isBefore(state.start, state.end);
103      var end = startBefore ? action.interval.end :
       ↪  action.interval.start;
104      var { start, direction } = state;
105      let expectedDirection = startBefore
106        ? selectionDirection.FORWARD
107        : selectionDirection.BACKWARD;
108      if (direction !== expectedDirection) {
109        direction = expectedDirection;
110        let interval = action.settings.interval * (startBefore ? -1 : 1);
111        start = toHourMinute(toMinutes(start) + interval);
112      }
113      let actualSchedule: DayInterval = {
114        start: startBefore ? start : end,
115        end: startBefore ? end : state.end,
116        dayOfWeek: state.dayOfWeek
117      };
118      let conflict = action.schedules.find(schedule =>
119        hasConflict(schedule, actualSchedule)
120      );
121      if (conflict || isEqual(end, state.end) || isEqual(end, start)) {
122        return state;
123      }
```

```
      return {
        ...state,
        start,
        end,
        direction
      };
  }
}

export function useCalendarSelect(options: CalendarSelectOptions) {
  const [selection, dispatch] = useReducer(selectionStateReducer, null);
  const [selected, setSelected] = useState<DayInterval | null>(null);
  let { onScheduleClick, onSelect, schedules } = options.props;
  let { settings } = options;
  const start = useCallback(
    function(interval: DayInterval) {
      if (!onSelect || selection !== null) {
        return;
      }
      let conflict = schedules.find(schedule =>
        hasConflict(schedule, interval)
      );
      if (!conflict) {
        dispatch({
          type: "start",
          interval
        });
      }
    },
    [selection, onScheduleClick, onSelect, schedules]
  );
  const onMouseDown = useCallback(
    function(evt: MouseEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let interval = getDayInterval(evt.currentTarget, evt.clientY,
      ↪  settings);
```

```
      if (interval) {
        start(interval);
      }
    },
    [start]
  );
  const onTouchStart = useCallback(
    function(evt: TouchEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let interval = getDayInterval(
        evt.currentTarget,
        evt.changedTouches[0].clientY,
        settings
      );
      if (interval) {
        start(interval);
        if (!onSelect) {
          setSelected(interval);
        }
      }
    },
    [start]
  );
  let move = useCallback(
    function(interval: DayInterval) {
      if (!onSelect || selection === null) {
        return;
      }
      dispatch({
        type: "move",
        interval,
        settings,
        schedules
      });
    },
    [selection, onSelect, settings, schedules]
```

```
  );
  const onMouseMove = useCallback(
    function(evt: MouseEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let schedule = getDayInterval(evt.currentTarget, evt.clientY,
      ↪  settings);
      if (schedule) {
        move(schedule);
        if (selection === null) {
          setSelected(schedule);
        }
      }
    },
    [move, onSelect, selection]
  );
  const onTouchMove = useCallback(
    function(evt: TouchEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let schedule = getDayInterval(
        evt.currentTarget,
        evt.changedTouches[0].clientY,
        settings
      );
      if (schedule) {
        move(schedule);
      }
    },
    [move, onSelect]
  );
  const end = useCallback(
    function(interval: DayInterval) {
      let schedule = schedules.find(schedule =>
        hasConflict(schedule, interval)
```

```
      );
      if (selection === null && schedule && onScheduleClick) {
        onScheduleClick(schedule);
      } else if (
        selection !== null &&
        selection.dayOfWeek === interval.dayOfWeek &&
        onSelect
      ) {
        let { start, end } = selection;
        let startBefore = isBefore(start, end);
        onSelect(startBefore ? start : end, startBefore ? end : start);
      }
      dispatch({ type: "clear" });
    },
    [onScheduleClick, onSelect, selection, schedules]
  );
  const onMouseUp = useCallback(
    function(evt: MouseEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let schedule = getDayInterval(evt.currentTarget, evt.clientY,
      ↪  settings);
      if (schedule) {
        end(schedule);
      }
    },
    [end, onSelect]
  );
  const onTouchEnd = useCallback(
    function(evt: TouchEvent<HTMLDivElement>) {
      evt.preventDefault();
      if (onSelect) {
        evt.stopPropagation();
      }
      let schedule = getDayInterval(
        evt.currentTarget,
        evt.changedTouches[0].clientY,
```

```
        settings
      );
      if (schedule) {
        end(schedule);
      }
    },
    [end, onSelect]
  );
  const onMouseOut = useCallback(function() {
    setSelected(null);
    dispatch({ type: "clear" });
  }, []);
  let newSelection = selection;
  if (
    newSelection !== null &&
    !isBefore(newSelection.start, newSelection.end)
  ) {
    newSelection = {
      ...newSelection,
      start: newSelection.end,
      end: newSelection.start
    };
  }
  return {
    selection: newSelection,
    selected,
    onMouseUp,
    onMouseMove,
    onMouseDown,
    onMouseOut,
    onTouchEnd,
    onTouchMove,
    onTouchStart
  };
}
```

Arquivo interactive.slide.tsx

```
import {
  useCallback,
```

```
  MouseEvent,
  TouchEvent,
  useState,
  useLayoutEffect
} from "react";
import { pickPosition, allWeekDays, allHours } from "./utilities";
import {
  CalendarSettings,
  Start,
  Dimension,
  MovementMode,
  Position
} from "./types";

type CalendarPositionProps = CalendarSettings & {
  visibleDays: number;
  slide: boolean;
  startHour: number;
  endHour: number;
};

interface State {
  column: number;
  columnPosition: number;
  row: number;
  rowPosition: number;
  factor: number;
  animate: boolean;
  start: Start | null;
  position: Position | null;
}

export function useCalendarPosition(
  props: CalendarPositionProps,
  containerDimension: Dimension | null
) {
  let [state, setState] = useState<State>({
    column: 0,
    columnPosition: 0,
```

```
42      row: 0,
43      rowPosition: 0,
44      factor: 0,
45      animate: false,
46      start: null,
47      position: null
48    });
49    let startMove = useCallback(
50      function(unified: Position) {
51        if (!props.slide) {
52          return;
53        }
54        let position = pickPosition(unified);
55        setState(oldState => {
56          return {
57            ...oldState,
58            animate: false,
59            start: {
60              x: position.clientX,
61              y: position.clientY,
62              mode: MovementMode.NONE
63            }
64          };
65        });
66      },
67      [props.slide]
68    );
69
70    let onMouseDown = useCallback(
71      function(evt: MouseEvent) {
72        evt.preventDefault();
73        startMove(evt);
74      },
75      [startMove]
76    );
77
78    let onTouchStart = useCallback(
79      function(evt: TouchEvent) {
80        evt.preventDefault();
```

```
      startMove(evt.changedTouches[0]);
    },
    [startMove]
  );

  let move = useCallback(
    function(unified: Position) {
      let position = pickPosition(unified);
      setState(oldState => {
        if (!oldState.start || containerDimension === null) {
          return oldState;
        }
        let newState: State = {
          ...oldState,
          columnPosition: position.clientX - oldState.start.x,
          rowPosition: position.clientY - oldState.start.y
        };
        let mode = oldState.start.mode;
        if (mode === MovementMode.NONE) {
          const { height, width } = containerDimension;
          const columnFactor = +(
            (Math.sign(newState.columnPosition) *
            ↪  newState.columnPosition) /
            width
          ).toFixed(2);
          const rowFactor = +(
            (Math.sign(newState.rowPosition) * newState.rowPosition) /
            height
          ).toFixed(2);
          if (columnFactor > 0.1 || rowFactor > 0.1) {
            mode =
              columnFactor > 0.1
                ? MovementMode.HORIZONTAL
                : MovementMode.VERTICAL;
            newState.start = {
              ...oldState.start,
              mode
            };
          }
```

```
        }
        newState.rowPosition = Math.round(newState.rowPosition);
        newState.columnPosition = Math.round(newState.columnPosition);
        if (mode === MovementMode.HORIZONTAL) {
          newState.rowPosition = 0;
        } else {
          newState.columnPosition = 0;
        }
        return newState;
      });
    },
    [containerDimension]
  );

  let onMouseMove = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      move(evt);
    },
    [move]
  );

  let onTouchMove = useCallback(
    function(evt: TouchEvent) {
      evt.preventDefault();
      move(evt.changedTouches[0]);
    },
    [move]
  );

  useLayoutEffect(() => {
    if (!state.animate) {
      return;
    }
    setState(oldState => {
      if (
        !oldState.start ||
        !oldState.position ||
        containerDimension === null
```

```
158       ) {
159         return oldState;
160       }
161       const newState: State = {
162         ...oldState,
163         start: null,
164         columnPosition: 0,
165         rowPosition: 0,
166         factor: 0,
167         position: null
168       };
169       if (oldState.start.mode === MovementMode.HORIZONTAL) {
170         const columnPosition = oldState.position.clientX -
          ↪  oldState.start.x;
171         const columnSign = Math.sign(columnPosition);
172         const { width } = containerDimension;
173         const totalDays = allWeekDays.length - props.visibleDays;
174         const step = Math.abs(Math.round(columnPosition / (width / 3)));
175         newState.factor = +((columnSign * columnPosition) /
          ↪  width).toFixed(2);
176         newState.column -= columnSign * step;
177         newState.column = Math.max(newState.column, 0);
178         newState.column = Math.min(newState.column, totalDays);
179       } else {
180         const rowPosition = oldState.position.clientY - oldState.start.y;
181         const rowSign = Math.sign(rowPosition);
182         const { startHour } = props;
183         const step = Math.abs(Math.round(rowPosition /
          ↪  props.hourHeight));
184         const { height } = containerDimension;
185         newState.factor = +((rowSign * rowPosition) / height).toFixed(2);
186         newState.row -= Math.sign(rowPosition) * step;
187         newState.row = Math.min(
188           newState.row,
189           allHours.length - height / props.hourHeight
190         );
191         newState.row = Math.max(newState.row, -startHour);
192       }
193       newState.factor = 1 - newState.factor;
```

```
      return newState;
    });
  }, [state.animate, props.startHour, props.visibleDays,
  ↪  props.hourHeight]);

  let endMove = useCallback(function(unified: Position) {
    let position = pickPosition(unified);
    setState(oldState => {
      return { ...oldState, animate: true, position };
    });
  }, []);

  let onMouseUp = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      endMove(evt);
    },
    [endMove]
  );

  let onTouchEnd = useCallback(
    function(evt: TouchEvent) {
      evt.preventDefault();
      endMove(evt.changedTouches[0]);
    },
    [endMove]
  );

  return {
    state,
    onMouseDown,
    onMouseMove,
    onMouseUp,
    onTouchStart,
    onTouchMove,
    onTouchEnd
  };
}
```

Arquivo interactive.stories.tsx

```
import { storiesOf } from "@storybook/react";
import React from "react";
import { CalendarInteractive } from "./interactive";
import { Schedule } from "./types";

const schedulesData: Schedule[] = [
  {
    start: {
      hour: 8,
      minute: 20
    },
    end: {
      hour: 10,
      minute: 10
    },
    dayOfWeek: 2,
    title: "A"
  },
  {
    start: {
      hour: 8,
      minute: 20
    },
    end: {
      hour: 11,
      minute: 50
    },
    dayOfWeek: 2,
    title: "B"
  },
  {
    start: {
      hour: 13,
      minute: 30
    },
    end: {
      hour: 15,
```

```
      minute: 10
    },
    dayOfWeek: 3,
    title: "C"
  },
  {
    start: {
      hour: 13,
      minute: 30
    },
    end: {
      hour: 14,
      minute: 0
    },
    dayOfWeek: 4,
    title: "D"
  },
  {
    start: {
      hour: 13,
      minute: 0
    },
    end: {
      hour: 13,
      minute: 30
    },
    dayOfWeek: 4,
    title: "INE5412"
  }
];

storiesOf("CalendarInteractive", module).add("empty", () => {
  return <CalendarInteractive schedules={schedulesData} />;
});
```

Arquivo interactive.tsx

```
import React, { ReactNode, Fragment, useRef, useMemo } from "react";
import {
  ScheduleUI,
```

```
  ScheduleType,
  Schedule,
  HourMinute,
  CalendarSettings,
  DayInterval
} from "./types";
import {
  formatHourMinute,
  allWeekDays,
  allHours,
  getEndHour,
  getStartHour,
  getSchedules,
  isScheduleEqual,
  getScheduleKey
} from "./utilities";
import { useContainerDimension } from "./interactive.dimension";
import { useCalendarPosition } from "./interactive.slide";
import { useCalendarSelect } from "./interactive.select";
import { useViewportHeight } from "./interactive.height";

function Cell(
  schedule: ScheduleUI,
  settings: CalendarSettings,
  selection: DayInterval | null
) {
  const { hourHeight } = settings;
  const style = {
    marginTop: schedule.marginTop + "px",
    height: schedule.height + "px"
  };
  let classes: string = "cell ";
  if (selection !== null && isScheduleEqual(schedule, selection)) {
    classes += "hover ";
  }
  if (schedule.height <= hourHeight) {
    classes += "event-small ";
  }
  let content: ReactNode;
```

```
43    let start = formatHourMinute(schedule.start);
44    let end = formatHourMinute(schedule.end);
45    if (schedule.type === ScheduleType.PLACEHOLDER) {
46      classes += "placeholder";
47    } else {
48      style["backgroundColor"] = schedule.color;
49      classes += "event";
50      content = (
51        <Fragment>
52          <span className="time">
53            {start} - {end}
54          </span>
55          <span className="title">{schedule.title}</span>
56        </Fragment>
57      );
58    }
59    if (schedule.start.minute === 0) {
60      classes += " item";
61    }
62    let key = getScheduleKey(schedule);
63    return (
64      <div className={classes} style={style} key={key}>
65        {content}
66      </div>
67    );
68  }
69
70  export type CalendarInteractiveProps = Partial<CalendarSettings> & {
71    visibleDays?: number;
72    slide?: boolean;
73    startHour?: number;
74    endHour?: number;
75    schedules: Schedule[];
76    temporarySelection?: Pick<Schedule, "title" | "color">;
77    onSelect?: (start: HourMinute, end: HourMinute) => void;
78    onScheduleClick?: (schedule: Schedule) => void;
79  };
80
81  function noop(s: string) {
```

```
  return s;
}

function crop(s: string) {
  return s.substr(0, 3);
}

export function CalendarInteractive(props: CalendarInteractiveProps) {
  let ref = useRef<HTMLDivElement>(null);
  let containerDimension = useContainerDimension(ref);
  let height = useViewportHeight(ref);
  let settings: CalendarSettings = useMemo(
    function() {
      let hourHeight = Math.max(props.hourHeight || 0, 30);
      return {
        hourHeight,
        interval: Math.max(props.interval || 0, 15 / (hourHeight / 60))
      };
    },
    [props.hourHeight, props.interval]
  );
  let { selection, selected, ...selectEvents } = useCalendarSelect({
    props,
    settings
  });
  const startHour = getStartHour(props.startHour);
  const endHour = getEndHour(props.endHour);
  let visibleDays = !props.visibleDays ? allWeekDays.length :
  ↪  props.visibleDays;
  let { state, ...positionEvents } = useCalendarPosition(
    {
      endHour,
      startHour,
      ...settings,
      visibleDays,
      slide: !!props.slide
    },
    containerDimension
  );
```

```
120   const data = useMemo(
121     function() {
122       let newSchedules = props.schedules;
123       if (selection !== null) {
124         newSchedules = newSchedules.concat([
125           {
126             ...selection,
127             color: props.temporarySelection?.color ?? "#ccc",
128             title: props.temporarySelection?.title ?? "Novo Evento"
129           }
130         ]);
131       }
132       return getSchedules(newSchedules, settings);
133     },
134     [props.schedules, settings, selection]
135   );
136   let classes = "better-calendar";
137   if (state.animate) {
138     classes += " animate";
139   }
140   let style = {
141     "--column": state.column,
142     "--columnPosition": state.columnPosition + "px",
143     "--row": state.row + startHour,
144     "--rowPosition": state.rowPosition + "px",
145     "--factor": state.factor,
146     "--hourHeight": settings.hourHeight + "px",
147     "--visibleHours": endHour - startHour,
148     "--totalColumns": allWeekDays.length,
149     "--hoverColor": props.temporarySelection?.color ?? "#ccc",
150     "--visibleColumns": visibleDays
151   } as React.CSSProperties;
152
153   if (height) {
154     style["height"] = height + "px";
155   }
156   let small = false;
157   if (containerDimension) {
```

```
158       // Size of the table without the first column (which contains the
          ↪  hours)
159       const viewport = containerDimension.width - 60;
160       small = viewport / visibleDays <= 420 / 7;
161     }
162     let renderWeekDay = small ? crop : noop;
163     return (
164       <div className={classes} style={style} ref={ref} {...positionEvents}>
165         <div id="header">
166           <div className="cell">Horário</div>
167           <div className="weekdays">
168             {allWeekDays.map(weekDay => {
169               return (
170                 <div className="column cell" key={weekDay}>
171                   {renderWeekDay(weekDay)}
172                 </div>
173               );
174             })}
175           </div>
176         </div>
177         <div id="content">
178           <div className="hours">
179             {allHours.map(hour => {
180               const hourMinute = formatHourMinute(hour);
181               return (
182                 <div className="cell item" key={hourMinute}>
183                   {hourMinute}
184                 </div>
185               );
186             })}
187           </div>
188           <div className="inner">
189             {allWeekDays.map((weekDay, i) => {
190               return (
191                 <div className="column" key={weekDay}>
192                   <div className="column-data">
193                     <div
194                       className="overlay"
195                       {...selectEvents}
```

```
                    data-day-of-week={data[i][0].dayOfWeek}
                  ></div>
                  <div className="data">
                    {data[i].map((schedule, j) =>
                      Cell(schedule, settings, selected)
                    )}
                  </div>
                </div>
              </div>
            );
          })}
        </div>
      </div>
    </div>
  );
}
```

Arquivo thumbnail.tsx

```
import React, { ReactNode, useContext } from "react";
import {
  allHours,
  getHeightDiff,
  allWeekDays,
  getStartHour,
  getEndHour,
  formatHourMinute,
  getDayOfWeekIndex,
  isBefore
} from "./utilities";
import { HourMinute, Schedule } from "./types";

interface CalendarContextValue {
  hourHeight: number;
  start: HourMinute;
  baseCanvas: HTMLCanvasElement;
  dayWidth: number;
  textColor: string;
}
```

```
21
22 const CalendarContext = React.createContext<CalendarContextValue |
   ↪  null>(null);
23
24 interface CalendarProviderProps {
25   width: number;
26   height: number;
27   numColumns?: number;
28   startHour?: number;
29   endHour?: number;
30   backgroundColor?: string;
31   strokeColor?: string;
32   textColor?: string;
33   children: ReactNode;
34 }
35
36 export function CalendarProvider(props: CalendarProviderProps) {
37   let {
38     children,
39     width,
40     height,
41     numColumns,
42     backgroundColor,
43     strokeColor,
44     textColor
45   } = props;
46   backgroundColor = backgroundColor === undefined ? "#fff" :
   ↪  backgroundColor;
47   strokeColor = strokeColor === undefined ? "#999" : strokeColor;
48   textColor = textColor === undefined ? "#000" : textColor;
49   numColumns = Math.max(Math.min(numColumns || 9, allWeekDays.length), 1)
   ↪  + 1;
50   let startHour = getStartHour(props.startHour);
51   let endHour = getEndHour(props.endHour);
52   const start: HourMinute = allHours[startHour];
53   const baseCanvas = document.createElement("canvas");
54   baseCanvas.width = width;
55   baseCanvas.height = height;
56   const ctx = baseCanvas.getContext("2d");
```

```
57    const dayWidth = width / numColumns;
58    const interval = endHour - startHour + 1;
59    const hourHeight = height / interval;
60    if (ctx) {
61      ctx.fillStyle = backgroundColor;
62      ctx.strokeStyle = strokeColor;
63      ctx.fillRect(0, 0, width, height);
64      ctx.font = "normal 8px Roboto";
65      ctx.textAlign = "center";
66      for (let j = 1; j < interval; j++) {
67        const y = (j + 1) * hourHeight;
68        const text = formatHourMinute({
69          hour: start.hour + j - 1,
70          minute: 0
71        });
72        ctx.fillStyle = textColor;
73        ctx.fillText(text, dayWidth / 2, y - hourHeight / 4);
74      }
75      for (let i = 0; i < numColumns; i++) {
76        const x = i * dayWidth;
77        const x2 = x + dayWidth;
78        ctx.beginPath();
79        ctx.strokeStyle = strokeColor;
80        ctx.moveTo(x, 0);
81        ctx.lineTo(x, height);
82        ctx.stroke();
83        const text = i > 0 ? allWeekDays[i - 1] : "Horário";
84        ctx.fillStyle = textColor;
85        ctx.fillText(text, x + dayWidth / 2, hourHeight - hourHeight / 4);
86        for (let j = 0; j < interval; j++) {
87          const y = (j + 1) * hourHeight;
88          ctx.beginPath();
89          ctx.strokeStyle = strokeColor;
90          ctx.moveTo(x, y);
91          ctx.lineTo(x2, y);
92          ctx.stroke();
93        }
94      }
95    } else {
```

```
    throw new Error("Cannot create 2D context for thumbnail provider");
  }
  let value: CalendarContextValue = {
    baseCanvas,
    start,
    hourHeight,
    dayWidth,
    textColor
  };
  return (
    <CalendarContext.Provider value={value}>
      {children}
    </CalendarContext.Provider>
  );
}

export interface ThumbnailProps {
  width: number;
  height: number;
  src: string;
}

export interface CalendarThumbnailProps {
  schedules: Schedule[];
  showTitle?: boolean;
  render?(props: ThumbnailProps): JSX.Element;
}

function renderThumbnail(props: ThumbnailProps) {
  return <img {...props} />;
}

export function CalendarThumbnail(props: CalendarThumbnailProps) {
  let context = useContext(CalendarContext);
  if (context === null) {
    throw new Error(
      "Cannot use CalendarThumbnail without a parent CalendarProvider"
    );
  }
```

```
135   const targetCanvas = document.createElement("canvas");
136   const { baseCanvas, dayWidth, hourHeight, start, textColor } = context;
137   const { width, height } = baseCanvas;
138   targetCanvas.width = width;
139   targetCanvas.height = height;
140   const ctx = targetCanvas.getContext("2d");
141
142   let { render, schedules, showTitle } = props;
143   if (render === undefined) {
144     render = renderThumbnail;
145   }
146   if (ctx) {
147     let image = (baseCanvas as unknown) as CanvasImageSource;
148     ctx.drawImage(image, 0, 0);
149     for (let schedule of schedules) {
150       if (isBefore(schedule.start, start)) {
151         continue;
152       }
153       let dayOfWeek = getDayOfWeekIndex(schedule.dayOfWeek);
154       let x = (dayOfWeek + 1) * dayWidth;
155       let y =
156         getHeightDiff(hourHeight, {
157           start,
158           end: schedule.start
159         }) + hourHeight;
160       let height = getHeightDiff(hourHeight, schedule);
161       ctx.fillStyle = schedule.color || "#ccc";
162       ctx.fillRect(x, y, dayWidth, height);
163
164       if (showTitle && schedule.title && height >= hourHeight) {
165         ctx.font = "normal 8px Roboto";
166         ctx.textAlign = "center";
167         ctx.fillStyle = textColor;
168         ctx.fillText(schedule.title, x + dayWidth / 2, y + hourHeight /
          ↪  2);
169       }
170     }
171   } else {
172     throw new Error("Cannot create 2D context for calendar thumbnail");
```

```
173   }
174   const src = targetCanvas.toDataURL();
175   const renderProps: ThumbnailProps = {
176     src,
177     width,
178     height
179   };
180   return render(renderProps);
181 }
```

Arquivo types.ts

```
1  export type HourMinute = {
2    hour: number;
3    minute: number;
4  };
5
6  export type DayHourMinute = HourMinute & {
7    dayOfWeek: number;
8  };
9
10 export type Interval = {
11   start: HourMinute;
12   end: HourMinute;
13 };
14
15 export type DayInterval = Interval & {
16   dayOfWeek: number;
17 };
18
19 export type Schedule = DayInterval & {
20   title?: string;
21   color?: string;
22   id?: string;
23 };
24
25 export type Position = {
26   clientX: number;
27   clientY: number;
28 };
```

```
29
30 export enum ScheduleType {
31   ITEM,
32   CONFLICT,
33   PLACEHOLDER
34 }
35
36 export const ScheduleTypeValue = ScheduleType;
37 export type ScheduleRecord = Schedule & {
38   type: ScheduleType;
39 };
40
41 export type ScheduleUI = ScheduleRecord & {
42   marginTop: number;
43   height: number;
44   color: string;
45 };
46
47 export type CalendarSettings = {
48   hourHeight: number;
49   interval: number;
50 };
51
52 export const enum MovementMode {
53   NONE,
54   VERTICAL,
55   HORIZONTAL
56 }
57
58 export type Start = {
59   x: number;
60   y: number;
61   mode: MovementMode;
62 };
63
64 export type Dimension = {
65   width: number;
66   height: number;
67 };
```

Arquivo utilities.ts

```typescript
import {
  HourMinute,
  Schedule,
  Interval,
  ScheduleUI,
  ScheduleRecord,
  ScheduleType,
  Position,
  CalendarSettings,
  DayInterval
} from "./types";

export function isEqual(i: HourMinute, j: HourMinute): boolean {
  return i.hour === j.hour && i.minute === j.minute;
}

export function isBefore(i: HourMinute, j: HourMinute): boolean {
  return i.hour != j.hour ? i.hour < j.hour : i.minute < j.minute;
}

export function hasConflict(i: DayInterval, j: DayInterval): boolean {
  if (i.dayOfWeek !== j.dayOfWeek) {
    return false;
  }
  if (isBefore(j.start, i.start)) {
    return hasConflict(j, i);
  }
  return isBefore(j.start, i.end) && !isBefore(j.end, i.start);
}

export function compareHourTime(i: HourMinute, j: HourMinute): -1 | 1 | 0
↪ {
  return isEqual(i, j) ? 0 : isBefore(i, j) ? -1 : 1;
}

export function compareSchedule(i: DayInterval, j: DayInterval): -1 | 1 |
↪  0 {
```

```
36    if (i.dayOfWeek != j.dayOfWeek) {
37      return i.dayOfWeek < j.dayOfWeek ? -1 : 1;
38    }
39    return compareHourTime(i.start, j.start);
40  }
41
42  export function isScheduleEqual(i: DayInterval, j: DayInterval): boolean
    ↪ {
43    return compareSchedule(i, j) === 0;
44  }
45
46  export function last<T>(arr: T[]): T {
47    return arr[arr.length - 1];
48  }
49
50  export function toMinutes(now: HourMinute): number {
51    return now.hour * 60 + now.minute;
52  }
53
54  export function toHourMinute(minutes: number): HourMinute {
55    return {
56      hour: Math.floor(minutes / 60),
57      minute: minutes % 60
58    };
59  }
60
61  export function getHeightDiff(
62    hourHeight: number,
63    { start, end }: Interval
64  ): number {
65    let result = end;
66    if (start !== null) {
67      result = toHourMinute(toMinutes(end) - toMinutes(start));
68    }
69    const hour = result.hour * hourHeight;
70    const minute = (hourHeight / 60) * result.minute;
71    return hour + minute;
72  }
73
```

```
74  export function padZero(n: number): string {
75    return (n + "").padStart(2, "0");
76  }
77
78  export function formatHourMinute(time: HourMinute): string {
79    return padZero(time.hour) + ":" + padZero(time.minute);
80  }
81
82  export function getDayOfWeekIndex(dayOfWeek: number): number {
83    return (dayOfWeek - 2) % 8;
84  }
85
86  export function formatDayOfWeek(dayOfWeek: number): string {
87    return allWeekDays[getDayOfWeekIndex(dayOfWeek)];
88  }
89
90  export function getScheduleKey(schedule: Schedule): string {
91    return (
92      schedule.dayOfWeek +
93      "-" +
94      formatHourMinute(schedule.start) +
95      "-" +
96      formatHourMinute(schedule.end)
97    );
98  }
99
100 export function getValidHour(hour: number): number {
101   return Math.max(Math.min(hour, allHours.length), 0);
102 }
103
104 export function getStartHour(startHour?: number): number {
105   return getValidHour(startHour === undefined ? 7 : startHour);
106 }
107
108 export function getEndHour(endHour?: number): number {
109   return getValidHour(endHour === undefined ? allHours.length : endHour);
110 }
111
112 export function pickPosition(position: Position): Position {
```

```
113   let { clientX, clientY } = position;
114   return { clientX, clientY };
115 }
116
117 export const allHours: Readonly<HourMinute>[] = [];
118 for (let hour = 0; hour < 24; hour++) {
119   allHours.push({ hour, minute: 0 });
120 }
121
122 export const allWeekDays = [
123   "Segunda",
124   "Terça",
125   "Quarta",
126   "Quinta",
127   "Sexta",
128   "Sábado",
129   "Domingo"
130 ];
131
132 function getPlaceholders(
133   schedule: Schedule,
134   settings: CalendarSettings
135 ): ScheduleUI[] {
136   const result: ScheduleUI[] = [];
137   let { start, end, dayOfWeek } = schedule;
138   let { hourHeight, interval } = settings;
139   let actualMinute = toMinutes(schedule.start);
140   const endMinute = toMinutes(schedule.end);
141   let marginTop = 0;
142   if (actualMinute % interval) {
143     actualMinute = Math.ceil(actualMinute / interval) * interval;
144     marginTop = getHeightDiff(hourHeight, {
145       start,
146       end: toHourMinute(actualMinute)
147     });
148   }
149   while (actualMinute + interval <= endMinute) {
150     const start = toHourMinute(actualMinute);
151     actualMinute += interval;
```

```
152       const end = toHourMinute(actualMinute);
153       result.push({
154         color: "transparent",
155         start,
156         end,
157         dayOfWeek,
158         marginTop,
159         height: getHeightDiff(hourHeight, { start, end }),
160         type: ScheduleType.PLACEHOLDER
161       });
162       marginTop = 0;
163     }
164     if (actualMinute + interval > endMinute && actualMinute < endMinute) {
165       const start = toHourMinute(actualMinute);
166       result.push({
167         color: "transparent",
168         start,
169         end,
170         dayOfWeek,
171         marginTop,
172         height: getHeightDiff(hourHeight, { start, end }),
173         type: ScheduleType.PLACEHOLDER
174       });
175     }
176     return result;
177   }
178
179   function aggregateHourTime(
180     old: HourMinute[],
181     actual: HourMinute
182   ): HourMinute[] {
183     if (!old.length || !isEqual(last(old), actual)) {
184       old.push(actual);
185     }
186     return old;
187   }
188
189   export function getSchedules(
190     schedules: Schedule[],
```

```
  settings: CalendarSettings
): ScheduleUI[][] {
  schedules = schedules.slice().sort(compareSchedule);
  const newSchedules: ScheduleRecord[][] = Array(allWeekDays.length)
    .fill(1)
    .map((): ScheduleRecord[] => []);
  for (const schedule of schedules) {
    if (isBefore(schedule.end, schedule.start)) {
      throw new Error("Invalid schedule");
    }
    const dayOfWeek = getDayOfWeekIndex(schedule.dayOfWeek);
    const list = newSchedules[dayOfWeek];
    if (list.length && hasConflict(last(list), schedule)) {
      // Has conflict! Need to process it in a appropriate way
      const lastSchedule = list.pop() as Schedule;
      const parts = [
        lastSchedule.start,
        lastSchedule.end,
        schedule.start,
        schedule.end
      ]
        .sort(compareHourTime)
        .reduce(aggregateHourTime, []);
      switch (parts.length) {
        case 4:
          list.push({
            ...(isEqual(schedule.start, parts[0]) ? schedule :
            ↪  lastSchedule),
            end: parts[1],
            type: ScheduleType.ITEM
          });
          list.push({
            start: parts[1],
            end: parts[2],
            dayOfWeek: schedule.dayOfWeek,
            type: ScheduleType.CONFLICT
          });
          list.push({
```

```
228             ...(isEqual(schedule.end, parts[3]) ? schedule :
                ↪ lastSchedule),
229             start: parts[2],
230             type: ScheduleType.ITEM
231           });
232           break;
233         case 3:
234           const scheduleStart = isEqual(parts[1], schedule.start);
235           list.push({
236             ...lastSchedule,
237             end: parts[1],
238             type: !scheduleStart ? ScheduleType.CONFLICT :
                ↪ ScheduleType.ITEM
239           });
240           list.push({
241             ...schedule,
242             start: parts[1],
243             type: scheduleStart ? ScheduleType.CONFLICT :
                ↪ ScheduleType.ITEM
244           });
245           break;
246         case 2:
247         case 1:
248           list.push({
249             ...lastSchedule,
250             type: ScheduleType.CONFLICT
251           });
252       }
253     } else {
254       // No conflict! Just push new schedule
255       list.push({
256         ...schedule,
257         type: ScheduleType.ITEM
258       });
259     }
260     newSchedules[dayOfWeek] = list;
261   }
262   const start: HourMinute = allHours[0];
263   const end: HourMinute = {
```

```
264       hour: last(allHours).hour + 1,
265       minute: 0
266     };
267     return newSchedules.map(
268       (list: ScheduleRecord[], dayOfWeek: number): ScheduleUI[] => {
269         dayOfWeek = dayOfWeek + 2;
270         const result: ScheduleUI[] = [];
271         let prev = start;
272         for (const schedule of list) {
273           let placeholders = getPlaceholders(
274             {
275               dayOfWeek,
276               start: prev,
277               end: schedule.start
278             },
279             settings
280           );
281           result.push(...placeholders);
282           if (placeholders.length) {
283             prev = last(placeholders).end;
284           }
285           let isConflict = schedule.type === ScheduleType.CONFLICT;
286           result.push({
287             ...schedule,
288             title: isConflict ? "CONFLITO!" : schedule.title,
289             color: isConflict ? "red" : schedule.color || "#ccc",
290             height: getHeightDiff(settings.hourHeight, schedule),
291             marginTop: getHeightDiff(settings.hourHeight, {
292               end: schedule.start,
293               start: prev
294             })
295           });
296           prev = schedule.end;
297         }
298         result.push(
299           ...getPlaceholders(
300             {
301               dayOfWeek,
302               start: prev,
```

```
            end
          },
          settings
        )
      );
      return result;
    }
  );
}
```

B.2.66 Pasta src/js/pages/base/components/cards

Arquivo actions.tsx

```tsx
import React from "react";
import {
  CardActions,
  CardActionButtons,
  CardActionButton,
  CardActionIcons,
  CardActionIcon
} from "@rmwc/card";

export type ActionSelectionProps = {
  selected?: boolean;
  onAction?: () => void;
};

export type ActionsProps = ActionSelectionProps & {
  enabled?: boolean;
  onDetails?: () => void;
  onDelete?: () => void;
};

export function Actions(props: ActionsProps) {
  let { enabled, onDelete, onDetails, onAction, selected } = props;
  return (
    <CardActions>
      <CardActionButtons>
        {onAction ? (
```

```
27            <CardActionButton
28              outlined
29              trailingIcon={
30                !onDelete
31                  ? "add"
32                  : enabled
33                  ? "check_circle_outline"
34                  : "check_circle"
35              }
36              onClick={onAction}
37            >
38              {!onDelete ? "Adicionar" : enabled ? "Desativar" : "Ativar"}
39            </CardActionButton>
40          ) : null}
41          {onDetails ? (
42            <CardActionButton outlined trailingIcon="details"
              ↪  onClick={onDetails}>
43              Detalhes
44            </CardActionButton>
45          ) : null}
46        </CardActionButtons>
47        <CardActionIcons>
48          {onDelete ? <CardActionIcon icon="delete" onClick={onDelete} /> :
            ↪  null}
49          {selected != null ? (
50            <CardActionIcon
51              checked={selected}
52              disabled
53              icon={selected ? "bookmark" : "bookmark_border"}
54            />
55          ) : null}
56        </CardActionIcons>
57      </CardActions>
58    );
59  }
60
61  export function ActionSelection({ onAction, selected }:
    ↪  ActionSelectionProps) {
62    return (
```

```
63      <CardActions>
64        <CardActionButtons>
65          <CardActionButton
66            outlined
67            trailingIcon={!selected ? "check_circle_outline" :
              ↪  "check_circle"}
68            onClick={onAction}
69          >
70            {selected ? "Desativar" : "Ativar"}
71          </CardActionButton>
72        </CardActionButtons>
73      </CardActions>
74    );
75  }
```

Arquivo combination.tsx

```
1  import React from "react";
2  import {
3    CardPrimaryAction,
4    Card,
5    CardActions,
6    CardActionButtons,
7    CardActionButton
8  } from "@rmwc/card";
9  import { Typography } from "@rmwc/typography";
10 import { Actions, ActionsProps } from "./actions";
11 import { Team } from "../../../plan-details/store/plan";
12 import { CalendarThumbnail, Schedule } from "../calendar";
13 import { renderThumbnail } from "./utilities";
14
15 type CombinationCardProps = {
16   teams: Pick<Team, "schedules" | "discipline">[];
17   offerColors: { [id: string]: string };
18   index: number;
19   onSelect: () => void;
20   selected: boolean;
21 };
22
```

```
23 export default function CombinationCard(props: CombinationCardProps) {
24   let { teams, index, offerColors, selected, onSelect } = props;
25   let schedules: Schedule[] = [];
26   for (let team of teams) {
27     let color = offerColors[team.discipline.id];
28     for (let schedule of team.schedules) {
29       let r = parseInt(color.substr(1, 2), 16);
30       let g = parseInt(color.substr(3, 2), 16);
31       let b = parseInt(color.substr(5, 2), 16);
32       schedules.push({
33         ...schedule,
34         color: `rgba(${r}, ${g}, ${b}, 0.5)`,
35         title: team.discipline.code
36       });
37     }
38   }
39   return (
40     <Card outlined>
41       <CardPrimaryAction style={{ padding: "0.5rem 1rem 1rem 1rem" }}>
42         <CalendarThumbnail
43           schedules={schedules}
44           render={renderThumbnail}
45           showTitle
46         />
47         <Typography use="overline">
48           {teams.length} turma
49           {teams.length > 1 ? "s" : ""}
50         </Typography>
51         <Typography use="headline5">Combinação {index}</Typography>
52       </CardPrimaryAction>
53       <CardActions>
54         <CardActionButtons>
55           <CardActionButton
56             outlined
57             trailingIcon={!selected ? "check_circle_outline" :
               ↪  "check_circle"}
58             onClick={onSelect}
59           >
60             {selected ? "Desativar" : "Ativar"}
```

```
            </CardActionButton>
          </CardActionButtons>
        </CardActions>
      </Card>
    );
}
```

Arquivo discipline-offer.tsx

```
import React from "react";
import { CardPrimaryAction, Card, CardMedia } from "@rmwc/card";
import { Typography } from "@rmwc/typography";
import {
  RemoteDisciplineOffer,
  RemoteTeam,
  RemoteCampus,
  RemoteSchedule
} from "../../../../graphql";
import { CalendarThumbnail, Schedule, ThumbnailProps } from
↪  "../calendar";
import { getLessUsedColor } from
↪  "../../../plan-details/store/plan/colors";
import { Actions, ActionsProps } from "./actions";
import { renderThumbnail, plural } from "./utilities";

type TeamCardModel = Pick<RemoteTeam, "id" | "code"> & {
  schedules: Pick<RemoteSchedule, "start" | "end" | "dayOfWeek">[];
};

export type DisciplineOfferCardModel = Pick<
  RemoteDisciplineOffer,
  "code" | "name" | "id"
> & {
  teams: TeamCardModel[];
  campus?: Pick<RemoteCampus, "id" | "name">;
};
type DisciplineOfferCardProps = {
  disciplineOffer: DisciplineOfferCardModel;
  colors?: string[];
```

```
} & ActionsProps;

export default function DisciplineOfferCard(props:
↪  DisciplineOfferCardProps) {
  let { disciplineOffer, colors: originalColors, ...actions } = props;
  let schedules: Schedule[] = [];
  let colors: string[] = [...(originalColors ?? [])];
  let { teams } = disciplineOffer;
  for (let team of teams) {
    let color = getLessUsedColor(colors);
    colors.push(color);
    for (let schedule of team.schedules) {
      let r = parseInt(color.substr(1, 2), 16);
      let g = parseInt(color.substr(3, 2), 16);
      let b = parseInt(color.substr(5, 2), 16);
      schedules.push({
        ...schedule,
        color: `rgba(${r}, ${g}, ${b}, 0.5)`,
        title: team.code
      });
    }
  }
  return (
    <Card outlined>
      <CardPrimaryAction
        style={{ padding: "0.5rem 1rem 1rem 1rem" }}
        onClick={actions.onDetails}
      >
        <CalendarThumbnail schedules={schedules} render={renderThumbnail}
        ↪  />
        <Typography use="overline">
          {disciplineOffer.campus
            ? "Campus " + disciplineOffer.campus.name + " | "
            : ""}
          {plural(teams.length, "turma", "turmas", "Nenhuma")}
        </Typography>
        <Typography use="headline5">
          {disciplineOffer.code} - {disciplineOffer.name}
        </Typography>
```

```
66        </CardPrimaryAction>
67        <Actions {...actions} />
68      </Card>
69    );
70  }
```

Arquivo discipline.tsx

```
1   import React from "react";
2   import {
3     CardActionIcon,
4     CardActionIcons,
5     CardPrimaryAction,
6     Card
7   } from "@rmwc/card";
8   import { Typography } from "@rmwc/typography";
9   import { Discipline } from "../../../plan-details/store/plan";
10  import { Actions, ActionsProps } from "./actions";
11
12  export type DisciplineCardModel = Pick<
13    Discipline,
14    "id" | "code" | "name" | "type" | "description" | "course"
15  >;
16
17  type DisciplineCardProps = {
18    discipline: DisciplineCardModel;
19  } & ActionsProps;
20
21  export default function DisciplineCard(props: DisciplineCardProps) {
22    let { discipline, ...actions } = props;
23    return (
24      <Card outlined>
25        <CardPrimaryAction
26          style={{ padding: "0.5rem 1rem 1rem 1rem" }}
27          onClick={actions.onDetails}
28        >
29          <Typography use="overline">{discipline.course.name}</Typography>
30          <Typography use="headline5">
31            {discipline.code} - {discipline.name}
```

```
32          </Typography>
33          <Typography use="subtitle1">{discipline.type}</Typography>
34          <Typography use="body2">{discipline.description}</Typography>
35        </CardPrimaryAction>
36        <Actions {...actions} />
37      </Card>
38    );
39  }
```

Arquivo plan-version.tsx

```
1  import { Card, CardPrimaryAction } from "@rmwc/card";
2
3  import React from "react";
4
5  import { redirect } from "../../store/querystring";
6
7  import { Typography } from "@rmwc/typography";
8  import { useDispatch } from "react-redux";
9  import {
10   RemotePlanVersion,
11   RemoteUniversity,
12   RemotePeriod
13 } from "../../../../graphql";
14 import { CalendarThumbnail, Schedule } from "../calendar";
15 import { renderThumbnail, plural, formatDate } from "./utilities";
16 import { Actions, ActionSelection, ActionSelectionProps } from
   ↪  "./actions";
17
18 export type PlanVersionCardProps = {
19   version: Pick<
20     RemotePlanVersion,
21     "savedAt" | "totalPossibilities" | "selectedSchedules"
22   >;
23 } & ActionSelectionProps;
24
25 export function PlanVersionCard({ version, ...events }:
   ↪  PlanVersionCardProps) {
26   let schedules: Schedule[] = [];
```

```
27    if (version) {
28      for (let schedule of version.selectedSchedules) {
29        schedules.push({
30          ...schedule,
31          id: schedule.offerID
32        });
33      }
34    }
35    return (
36      <Card>
37        <CardPrimaryAction
38          onClick={events.onAction}
39          style={{ padding: "0 1rem 1rem 1rem" }}
40        >
41          <CalendarThumbnail
42            schedules={schedules}
43            render={renderThumbnail}
44            showTitle
45          />
46          <Typography use="headline6" tag="h2">
47            {formatDate(version.savedAt)}
48          </Typography>
49          <Typography use="body1" tag="div"
            ↪  theme="textSecondaryOnBackground">
50            {plural(
51              version.totalPossibilities,
52              "possibilidade",
53              "possibilidades",
54              "Nenhuma"
55            )}
56          </Typography>
57        </CardPrimaryAction>
58        <ActionSelection {...events} />
59      </Card>
60    );
61  }
```

Arquivo plan.tsx

```tsx
import { Card, CardPrimaryAction } from "@rmwc/card";

import React from "react";

import { redirect } from "../../store/querystring";

import { Typography } from "@rmwc/typography";
import { useDispatch } from "react-redux";
import {
  RemotePlan,
  RemoteUniversity,
  RemotePeriod,
  RemotePlanVersion
} from "../../../../graphql";
import { CalendarThumbnail, Schedule } from "../calendar";
import { renderThumbnail, plural, formatDate } from "./utilities";
import { Actions } from "./actions";

export type PlanCardProps = {
  plan: Pick<RemotePlan, "id" | "name" | "createdAt" | "lastModified"> &
  ↪ {
    university: Pick<RemoteUniversity, "id" | "name">;
    period: Pick<RemotePeriod, "id" | "name">;
  };
  version?: Pick<RemotePlanVersion, "totalPossibilities" |
  ↪  "selectedSchedules">;
  onDetails: () => void;
};

export function PlanCard({ plan, version, onDetails }: PlanCardProps) {
  let schedules: Schedule[] = [];
  if (version) {
    for (let schedule of version.selectedSchedules) {
      schedules.push({
        ...schedule,
        id: schedule.offerID
      });
    }
  }
```

```
38    return (
39      <Card>
40        <CardPrimaryAction
41          onClick={onDetails}
42          style={{ padding: "0 1rem 1rem 1rem" }}
43        >
44          <CalendarThumbnail
45            schedules={schedules}
46            render={renderThumbnail}
47            showTitle
48          />
49          <Typography use="headline6" tag="h2">
50            {plan.name}
51          </Typography>
52          <Typography
53            use="subtitle2"
54            tag="h3"
55            theme="textSecondaryOnBackground"
56            style={{ marginTop: "-1rem" }}
57          >
58            {plan.university.name} | {plan.period.name}
59          </Typography>
60          <Typography use="body1" tag="div"
            ↪  theme="textSecondaryOnBackground">
61            Criado em {formatDate(plan.createdAt)} | Última modificação
              ↪  em{" "}
62            {formatDate(plan.lastModified)}{" "}
63            {version
64              ? "| " +
65                plural(
66                  version.totalPossibilities,
67                  "possibilidade",
68                  "possibilidades",
69                  "Nenhuma"
70                )
71              : null}
72          </Typography>
73        </CardPrimaryAction>
74        <Actions onDetails={onDetails} />
```

```
75      </Card>
76    );
77  }
```

Arquivo teacher.tsx

```tsx
import React from "react";
import { CardPrimaryAction, Card, CardMedia } from "@rmwc/card";
import { Typography } from "@rmwc/typography";
import {
  RemoteTeam,
  RemoteCampus,
  RemoteSchedule,
  RemoteTeacher,
  RemoteDisciplineOffer
} from "../../../../graphql";
import { CalendarThumbnail, Schedule, ThumbnailProps } from
↪  "../calendar";
import { getLessUsedColor } from
↪  "../../../plan-details/store/plan/colors";
import { Actions, ActionsProps } from "./actions";
import { renderThumbnail, plural } from "./utilities";

type TeamCardModel = Pick<RemoteTeam, "id" | "code"> & {
  schedules: Pick<RemoteSchedule, "start" | "end" | "dayOfWeek">[];
  discipline: Pick<RemoteDisciplineOffer, "id" | "code">;
};

export type TeacherCardModel = Pick<RemoteTeacher, "name" | "id"> & {
  teams: TeamCardModel[];
  campus?: Pick<RemoteCampus, "id" | "name">;
};
type TeacherCardProps = {
  teacher: TeacherCardModel;
  colors?: string[];
} & ActionsProps;

export default function TeacherCard(props: TeacherCardProps) {
  let { teacher, colors: originalColors, ...actions } = props;
```

```
  let schedules: Schedule[] = [];
  let colors: string[] = [...(originalColors ?? [])];
  let { teams } = teacher;
  for (let team of teams) {
    let color = getLessUsedColor(colors);
    colors.push(color);
    for (let schedule of team.schedules) {
      let r = parseInt(color.substr(1, 2), 16);
      let g = parseInt(color.substr(3, 2), 16);
      let b = parseInt(color.substr(5, 2), 16);
      schedules.push({
        ...schedule,
        color: `rgba(${r}, ${g}, ${b}, 0.5)`,
        title: team.discipline.code
      });
    }
  }
  return (
    <Card outlined>
      <CardPrimaryAction
        style={{ padding: "0.5rem 1rem 1rem 1rem" }}
        onClick={actions.onDetails}
      >
        <CalendarThumbnail
          schedules={schedules}
          render={renderThumbnail}
          showTitle
        />
        <Typography use="overline">
          {plural(teams.length, "turma", "turmas", "Nenhuma")}
        </Typography>
        <Typography use="headline5">{teacher.name}</Typography>
      </CardPrimaryAction>
      <Actions {...actions} />
    </Card>
  );
}
```

Arquivo team.tsx

```
import React from "react";
import { CardPrimaryAction, Card, CardMedia } from "@rmwc/card";
import { Typography } from "@rmwc/typography";
import { CalendarThumbnail, ThumbnailProps, Schedule } from
  ↪  "../calendar";
import {
  RemoteTeam,
  RemoteDisciplineOffer,
  RemoteCampus
} from "../../../../graphql";
import { Actions, ActionsProps } from "./actions";
import { renderThumbnail } from "./utilities";

export type TeamCardModel = Pick<RemoteTeam, "id" | "code"> & {
  schedules: Schedule[];
  vacancies?: RemoteTeam["vacancies"];
  discipline: Pick<RemoteDisciplineOffer, "id" | "code" | "name">;
  campus?: Pick<RemoteCampus, "id" | "name">;
};

type TeamProps = {
  team: TeamCardModel;
  color?: string;
} & ActionsProps;

function TeamVacanciesCard({ team }: Pick<TeamProps, "team">) {
  if (!team.vacancies) {
    return null;
  }
  return (
    <Typography use="subtitle1">
      {team.vacancies.filled} vagas preenchidas/{team.vacancies.offered}
      ↪  vagas
      totais
    </Typography>
  );
}

```

```
export default function TeamCard({ team, color, ...actions }: TeamProps) {
  let schedules: Schedule[] = team.schedules;
  if (color) {
    schedules = schedules.map(schedule => {
      return { ...schedule, color };
    });
  }
  return (
    <Card outlined>
      <CardPrimaryAction
        style={{ padding: "0.5rem 1rem 1rem 1rem" }}
        onClick={actions.onDetails}
      >
        <CalendarThumbnail schedules={schedules} render={renderThumbnail} />
        <Typography use="overline">
          {team.discipline.code} - {team.discipline.name}{" "}
          {team.campus ? "| " + team.campus.name : ""}
        </Typography>
        <Typography use="headline5">{team.code}</Typography>
        <TeamVacanciesCard team={team} />
      </CardPrimaryAction>
      <Actions {...actions} />
    </Card>
  );
}
```

Arquivo utilities.tsx

```
import React from "react";
import { CardMedia } from "@rmwc/card";
import { ThumbnailProps } from "../calendar";

export function renderThumbnail(props: ThumbnailProps) {
  return (
    <CardMedia
      sixteenByNine
      style={{
```

```
        backgroundImage: "url(" + props.src + ")"
      }}
    />
  );
}

export function plural(
  count: number,
  singular: string,
  plural: string,
  none: string = "Nenhum"
) {
  let firstPart: string = "" + count;
  let secondPart = singular;
  if (count === 0) {
    firstPart = none;
  } else if (count > 1) {
    secondPart = plural;
  }
  return firstPart + " " + secondPart;
}

export function formatDate(date: Date) {
  return new Date(date).toLocaleDateString("pt-br", {
    day: "numeric",
    weekday: "long",
    year: "numeric",
    month: "numeric",
    hour: "numeric",
    minute: "numeric"
  });
}
```

B.2.67 Pasta src/js/pages/base/components/partials

Arquivo drawer.tsx

```
import React, { ReactNode, useCallback, Fragment } from "react";
import {
  List,
```

```
4     ListItem,
5     ListItemText,
6     ListItemGraphic,
7     ListDivider,
8     ListGroupSubheader
9   } from "@rmwc/list";
10  import {
11    Drawer,
12    DrawerHeader,
13    DrawerContent,
14    DrawerTitle,
15    DrawerSubtitle
16  } from "@rmwc/drawer";
17  import { useSelector, useDispatch } from "react-redux";
18  import { drawerClose, isDrawerOpen } from "../../store/ui";
19  import { getAuthenticatedUser, logout } from "../../store/auth";
20  import { push, getLocation, RouterRootState } from
    ↪  "connected-react-router";
21  import { useLoginCallback, useSignupCallback } from "../auth";
22
23  export interface ItemProps {
24    id: string;
25    selected?: string;
26    children: ReactNode;
27    onClick?(): void;
28  }
29
30  export function Item(props: ItemProps) {
31    let { selected, children, id, onClick: oldOnClick } = props;
32    let dispatch = useDispatch();
33    let onClick = useCallback(() => {
34      dispatch(drawerClose());
35      if (oldOnClick) {
36        oldOnClick();
37      }
38    }, [dispatch, oldOnClick]);
39    return (
40      <ListItem selected={selected === id} id={id} onClick={onClick}>
41        {children}
```

```
42      </ListItem>
43    );
44  }
45
46  interface Props {
47    selected: string;
48    children?: ReactNode;
49  }
50
51  function usePush(url: string): () => void {
52    let dispatch = useDispatch();
53    return useCallback(() => dispatch(push(url)), [dispatch]);
54  }
55
56  export default function AppDrawer(props: Props) {
57    let drawerOpen = useSelector(isDrawerOpen);
58    let dispatch = useDispatch();
59    let drawerCloseCallback = useCallback(() => dispatch(drawerClose()), [
60      dispatch
61    ]);
62    let logoutCallback = useCallback(() => dispatch(logout()), [dispatch]);
63    let universitiesCallback = usePush("/universidades/");
64    let planCallback = usePush("/planos/");
65    let accountCallback = usePush("/conta/");
66    let offlineCallback = usePush("/offline/");
67    const loginCallback = useLoginCallback();
68    const signupCallback = useSignupCallback();
69    let auth = useSelector(getAuthenticatedUser);
70    let { children, selected } = props;
71    return (
72      <Drawer
73        modal
74        open={drawerOpen}
75        key={drawerOpen + ""}
76        onClose={drawerCloseCallback}
77      >
78        <DrawerHeader>
79          <DrawerTitle>Guru da Matrícula</DrawerTitle>
```

```
80          <DrawerSubtitle>{auth ? auth.email : "Não
            ↪  autenticado"}</DrawerSubtitle>
81        </DrawerHeader>
82        <DrawerContent>
83          {children}
84          <ListGroupSubheader tag="h2">Conta</ListGroupSubheader>
85          <List tag="nav">
86            <Item id="plans" selected={selected} onClick={planCallback}>
87              <ListItemGraphic icon="folder" />
88              <ListItemText>Planos</ListItemText>
89            </Item>
90            <Item selected={selected} id="offline"
              ↪  onClick={offlineCallback}>
91              <ListItemGraphic icon="offline_pin" />
92              <ListItemText>Off-line</ListItemText>
93            </Item>
94            <Item selected={selected} id="account"
              ↪  onClick={accountCallback}>
95              <ListItemGraphic icon="person" />
96              <ListItemText>Conta</ListItemText>
97            </Item>
98            <Item
99              selected={selected}
100             id="universities"
101             onClick={universitiesCallback}
102           >
103             <ListItemGraphic icon="school" />
104             <ListItemText>Universidades</ListItemText>
105           </Item>
106           {auth ? (
107             <Item selected={selected} id="logout"
              ↪  onClick={logoutCallback}>
108               <ListItemGraphic icon="exit_to_app" />
109               <ListItemText>Sair</ListItemText>
110             </Item>
111           ) : (
112             <Fragment>
113               <Item selected={selected} id="login"
                ↪  onClick={loginCallback}>
```

```
114                 <ListItemGraphic icon="account_circle" />
115                 <ListItemText>Entrar</ListItemText>
116               </Item>
117               <Item selected={selected} id="signup"
                  ↪  onClick={signupCallback}>
118                 <ListItemGraphic icon="group_add" />
119                 <ListItemText>Cadastro</ListItemText>
120               </Item>
121             </Fragment>
122           )}
123         </List>
124       </DrawerContent>
125     </Drawer>
126   );
127 }
```

Arquivo responsive.tsx

```
1  import React, { ReactNode, useState, useEffect, Fragment } from "react";
2
3  interface Props {
4    children: ReactNode;
5  }
6
7  interface MatchState {
8    matches: boolean;
9  }
10
11 function useMediaQuery(query: string) {
12   const [matches, setMatch] = useState(false);
13   useEffect(() => {
14     let updateMatch = (evt: MatchState) => {
15       setMatch(evt.matches);
16     };
17     let handle = window.matchMedia(query);
18     handle.addListener(updateMatch);
19     updateMatch(handle);
20     return () => {
21       handle.removeListener(updateMatch);
```

```
    };
  }, [query]);
  return matches;
}

export function DesktopOnly(props: Props) {
  const matches = useMediaQuery("(min-width: 840px)");
  return matches ? <Fragment>{props.children}</Fragment> : null;
}

export function TabletOnly(props: Props) {
  const matches = useMediaQuery("(min-width: 480px) and (max-width:
  ↪  839px)");
  return matches ? <Fragment>{props.children}</Fragment> : null;
}

export function TabletDesktop(props: Props) {
  const matches = useMediaQuery("(min-width: 480px)");
  return matches ? <Fragment>{props.children}</Fragment> : null;
}

export function PhoneOnly(props: Props) {
  const matches = useMediaQuery("(max-width: 479px)");
  return matches ? <Fragment>{props.children}</Fragment> : null;
}
```

Arquivo top-bar.tsx

```
import React, { useCallback, ReactNode } from "react";
import {
  TopAppBar,
  TopAppBarFixedAdjust,
  TopAppBarRow,
  TopAppBarTitle,
  TopAppBarNavigationIcon,
  TopAppBarSection
} from "@rmwc/top-app-bar";
import { useDispatch } from "react-redux";
import { drawerToggle } from "../../store/ui";
```

```
12
13 interface Props {
14   title: string;
15   onTitleClick?(event?: Event): void;
16   show?: boolean;
17   children?: ReactNode;
18 }
19
20 export default function TopBar(props: Props) {
21   let { onTitleClick, children, title, show } = props;
22   let topAppBarClass = "top-app " + (show === false ? "hide" : "");
23   let dispatch = useDispatch();
24   let drawerToggleCallback = useCallback(() => {
25     dispatch(drawerToggle());
26   }, [dispatch]);
27   let titleProps = {};
28   if (onTitleClick) {
29     titleProps = {
30       onClick: onTitleClick,
31       role: "button",
32       tabIndex: 0
33     };
34   }
35   return (
36     <TopAppBar className={topAppBarClass} fixed>
37       <TopAppBarRow>
38         <TopAppBarSection alignStart>
39           <TopAppBarNavigationIcon icon="menu"
     ↪  onClick={drawerToggleCallback} />
40           <TopAppBarTitle {...titleProps}>{title}</TopAppBarTitle>
41         </TopAppBarSection>
42         {children}
43       </TopAppBarRow>
44     </TopAppBar>
45   );
46 }
47
48 export const TopBarContent = TopAppBarFixedAdjust;
```

B.2.68 Pasta src/js/pages/base/store/auth

Arquivo index.ts

```
export * from "./slice";
export * from "./selectors";
import getCSRFReduxModule, { FullState as CSRFFullState } from "../csrf";
import { reducer as auth } from "./slice";
import saga from "./sagas";
export { default as saga } from "./sagas";
import { FullState } from "./types";
import { ISagaModule } from "redux-dynamic-modules-saga";
export { SliceState, FullState } from "./types";
import client from "../../../../client";

export default function getReduxModule(): ISagaModule<
  FullState | CSRFFullState
>[] {
  return [
    getCSRFReduxModule(),
    {
      id: "auth",
      reducerMap: {
        auth
      },
      sagas: [
        {
          argument: { client },
          saga
        }
      ]
    }
  ];
}
```

Arquivo reducers.ts

```
import { SliceState, AuthStatus, User, LoginData } from "./types";
import { PayloadAction } from "@reduxjs/toolkit";

```

```
export function update(state: SliceState) {
  state.status = AuthStatus.LOADING;
}

export function logged(state: SliceState, { payload }:
↪  PayloadAction<User>) {
  state.status = AuthStatus.LOGGED;
  state.authenticatedUser = payload;
}

export function error(state: SliceState, { payload }:
↪  PayloadAction<string>) {
  state.authenticatedUser = null;
  state.status = AuthStatus.ERROR;
  state.errorMessage = payload;
}

export function loggedOut(state: SliceState) {
  state.authenticatedUser = null;
  state.status = AuthStatus.UNAUTHENTICATED;
}
```

Arquivo remote.ts

```
import query from "../../queries/user.gql";
import { ApolloClient, ApolloQueryResult } from "apollo-client";
import { RemoteGetUserQuery } from "../../../../graphql";

export async function getAuthenticatedUser<CacheShape>(
  client: ApolloClient<CacheShape>
): Promise<ApolloQueryResult<RemoteGetUserQuery>> {
  return client.query({
    query,
    fetchPolicy: "network-only"
  });
}
```

Arquivo sagas.ts

```
import {
  takeEvery,
  put,
  call,
  take,
  select,
  cancelled
} from "redux-saga/effects";
import { BASE_BACKEND_URL } from "../../../../config";
import { APIResponse } from "./types";
import {
  update as updateAction,
  error as errorAction,
  logged as loggedAction,
  logout as logoutAction,
  loggedOut as loggedOutAction
} from "./slice";
import { getAuthenticatedUser } from "./remote";
import { getAuthenticatedUser as getAuthenticatedUserSelector } from
↪  "./selectors";
import { fetchWithCSRFToken } from "../csrf";
import ApolloClient, { ApolloQueryResult } from "apollo-client";
import { channel, Channel } from "redux-saga";
import { RemoteGetUserQuery } from "../../../../graphql";

const authKey = "gdm_auth";

export function* update<CacheShape>(client: ApolloClient<CacheShape>) {
  let query: ApolloQueryResult<RemoteGetUserQuery> = yield call(
    getAuthenticatedUser,
    client
  );
  if (query.errors) {
    yield put(errorAction("Error while loading authentication status"));
  }
  let oldUser = yield select(getAuthenticatedUserSelector);
  if (
    query.data.loggedUser &&
    (!oldUser || query.data.loggedUser.id !== oldUser.id)
```

```
39    ) {
40      yield put(loggedAction(query.data.loggedUser));
41    } else if (!query.data.loggedUser && oldUser !== null) {
42      yield put(loggedOutAction());
43    }
44  }
45
46  export function* logout<CacheShape>(client: ApolloClient<CacheShape>) {
47    try {
48      let response: Response = yield call(
49        fetchWithCSRFToken,
50        `${BASE_BACKEND_URL}/auth/logout`,
51        {
52          body: "{}",
53          method: "POST"
54        }
55      );
56      let succeed = false;
57      if (response.status === 200) {
58        var data: APIResponse = yield call([response, response.json]);
59        succeed = data.status === "success";
60      }
61      if (succeed) {
62        yield put(loggedOutAction());
63        yield call(broadcast, client);
64      } else {
65        yield put(
66          errorAction(
67            "Erro desconhecido durante o logout. Tente novamente mais
              ↪ tarde."
68          )
69        );
70      }
71    } catch (e) {
72      yield put(errorAction("Unrecognized error happened on logout"));
73    }
74  }
75
76  export function* monitor() {
```

```
  let chan: Channel<string | null> = yield call(channel);
  function onStorage(event: StorageEvent) {
    if (event.key !== authKey) {
      return;
    }
    chan.put(event.newValue);
  }
  const eventName = "storage";
  window.addEventListener(eventName, onStorage, false);
  try {
    let stop = false;
    while (!stop) {
      let message = yield take(chan);
      let user = yield select(getAuthenticatedUserSelector);
      if (
        (message === null && user !== null) ||
        (user === null && message !== null)
      ) {
        window.location.reload();
      }
      stop = yield cancelled();
    }
  } finally {
    yield call([chan, chan.close]);
    window.removeEventListener(eventName, onStorage, false);
  }
}

export function* broadcast<CacheShape>(client: ApolloClient<CacheShape>)
↪ {
  let user = yield select(getAuthenticatedUserSelector);
  if (user) {
    localStorage.setItem(authKey, user.id);
  } else {
    localStorage.removeItem(authKey);
  }
}

```

```
114 // export function* clearStore<CacheShape>(client:
    ↪  ApolloClient<CacheShape>) {
115 //   client.resetStore();
116 // }
117
118 interface Options<CacheShape> {
119   client: ApolloClient<CacheShape>;
120 }
121
122 export default function* authSaga<CacheShape>({ client }:
    ↪  Options<CacheShape>) {
123   yield takeEvery(updateAction, update, client);
124   yield takeEvery(logoutAction, logout, client);
125   yield takeEvery(loggedAction, broadcast, client);
126   // yield takeEvery(loggedOutAction, clearStore, client);
127   //yield takeEvery(loggedAction, clearStore, client);
128
129   yield call(update, client);
130   yield call(monitor);
131 }
```

Arquivo selectors.ts

```
1  import { FullState, AuthStatus, SliceState, User } from "./types";
2
3  export function getSlice(state: FullState): SliceState {
4    return state.auth;
5  }
6
7  export function getStatus(state: FullState): AuthStatus {
8    return getSlice(state).status;
9  }
10
11 export function getAuthenticatedUser(state: FullState): User | null {
12   return getSlice(state).authenticatedUser;
13 }
14
15 export function getErrorMessage(state: FullState): string | undefined {
16   return getSlice(state).errorMessage;
17 }
```

Arquivo slice.ts

```
import { createSlice, createAction } from "@reduxjs/toolkit";
import { SliceState, AuthStatus, CreateAccountData } from "./types";
import * as reducers from "./reducers";

let authSlice = createSlice({
  name: "@@auth",
  initialState: {
    status: AuthStatus.UNAUTHENTICATED,
    authenticatedUser: null
  } as SliceState,
  reducers
});

const { actions, reducer } = authSlice;
export const { update, logged, error, loggedOut } = actions;
export const createAccount = createAction<CreateAccountData>(
  "@@auth/createAccount"
);
export const logout = createAction("@@auth/logout");
export { reducer };
```

Arquivo types.ts

```
export enum AuthStatus {
  LOADING,
  UNAUTHENTICATED,
  LOGGED,
  ERROR
}

export type User = {
  id: string;
  name: string;
  email: string;
  provider: string;
};

export type SliceState = {
```

```
  authenticatedUser: User | null;
  status: AuthStatus;
  errorMessage?: string;
};

export type FullState = {
  auth: SliceState;
};

export type LoginData = {
  email: string;
  password: string;
};

export type CreateAccountData = LoginData & {
  confirm_password: string;
};

export type APIResponse = {
  status: "success" | "failure";
  error?: string;
};
```

B.2.69 Pasta src/js/pages/base/store/csrf

Arquivo index.ts

```
export * from "./slice";
import { reducer as csrf } from "./slice";
import saga from "./sagas";
export * from "./sagas";
export { default as saga } from "./sagas";
import { FullState } from "./types";
import { ISagaModule } from "redux-dynamic-modules-saga";
export { SliceState, FullState } from "./types";

export default function getReduxModule(): ISagaModule<FullState> {
  return {
    id: "csrf",
    reducerMap: {
```

```
14        csrf
15      },
16      sagas: [saga]
17    };
18  }
```

Arquivo reducers.ts

```
1  import { SliceState, Status } from "./types";
2  import { PayloadAction } from "@reduxjs/toolkit";
3
4  export function update(state: SliceState) {
5    state.status = Status.LOADING;
6    state.value = undefined;
7  }
8
9  export function updated(state: SliceState, action: PayloadAction<string>)
   ↪ {
10   state.status = Status.LOADED;
11   state.value = action.payload;
12 }
13
14 export function error(state: SliceState) {
15   state.status = Status.ERROR;
16   state.value = undefined;
17 }
```

Arquivo sagas.ts

```
1  import {
2    update as updateAction,
3    updated as updatedAction,
4    error
5  } from "./slice";
6  import { takeEvery, put, call, select, take } from "redux-saga/effects";
7  import { BASE_BACKEND_URL } from "../../../../config";
8  import { getCSRFToken } from "./selectors";
9  import { PayloadAction } from "@reduxjs/toolkit";
10
```

```
export function* fetchWithCSRFToken(input: RequestInfo, init?:
↪  RequestInit) {
  if (!init) {
    init = {};
  }
  if (!init.headers) {
    init.headers = {};
  }
  let csrfToken = yield select(getCSRFToken);
  while (!csrfToken) {
    let action: PayloadAction<string> = yield take(updatedAction);
    csrfToken = action.payload;
  }
  init.credentials = "include";
  init.headers["X-CSRF-Token"] = csrfToken;
  let response: Response = yield call(fetch, input, init);
  return response;
}

export function* updateSaga() {
  let response: Response = yield call(fetch,
  ↪  `${BASE_BACKEND_URL}/api/csrf`, {
    credentials: "include"
  });
  if (response.status === 200) {
    let csrfToken: string = yield call([response, response.text]);
    yield put(updatedAction(csrfToken));
  } else {
    yield put(error());
  }
}

export default function* csrfSaga() {
  yield takeEvery(updateAction, updateSaga);
  yield call(updateSaga);
}
```

Arquivo selectors.ts

```
import { FullState, Status, SliceState } from "./types";
```

```
2
3  export function getSlice(state: FullState): SliceState {
4    return state.csrf;
5  }
6
7  export function getStatus(state: FullState): Status {
8    return getSlice(state).status;
9  }
10
11 export function getCSRFToken(state: FullState): string | null {
12   let { status, value } = getSlice(state);
13   if (status !== Status.LOADED || !value) {
14     return null;
15   }
16   return value;
17 }
```

Arquivo slice.ts

```
1  import { createSlice } from "@reduxjs/toolkit";
2  import { SliceState, Status } from "./types";
3  import * as reducers from "./reducers";
4
5  let csrfSlice = createSlice({
6    name: "@@csrf",
7    initialState: {
8      status: Status.LOADING
9    } as SliceState,
10   reducers
11 });
12
13 const { actions, reducer } = csrfSlice;
14 export const { update, updated, error } = actions;
15 export { reducer };
```

Arquivo types.ts

```
1  export enum Status {
2    LOADING,
```

```
  LOADED,
  ERROR
}

export type SliceState = {
  value?: string;
  status: Status;
};

export type FullState = {
  csrf: SliceState;
};
```

B.2.70 Pasta src/js/pages/base/store/querystring

Arquivo actions.ts

```
import { createAction } from "@reduxjs/toolkit";
import { QueryString, URLQueryString } from "./types";
import { stringify } from "./utilities";
import { push } from "connected-react-router";

export const pushQueryString =
↪  createAction<QueryString>("@@querystring/push");

export function redirect(payload: URLQueryString) {
  let newSearch = stringify(payload.querystring);
  let link = payload.url + newSearch;
  return push(link);
}
```

Arquivo index.ts

```
import { ISagaModule } from "redux-dynamic-modules-saga";

export * from "./actions";
export * from "./types";
export * from "./selectors";
export * from "./sagas";
export * from "./utilities";
```

```
8  import saga from "./sagas";
9
10 export default function getReduxModule(): ISagaModule<never> {
11   return {
12     id: "querystring",
13     sagas: [saga]
14   };
15 }
```

Arquivo sagas.ts

```
1  import { takeEvery, select, put, call } from "redux-saga/effects";
2  import { QueryString, URLQueryString } from "./types";
3  import { PayloadAction } from "@reduxjs/toolkit";
4  import { push, getLocation, CALL_HISTORY_METHOD } from
   ↪  "connected-react-router";
5  import { getQueryString, getLinkFromData } from "./selectors";
6  import { pushQueryString, redirect } from "./actions";
7  import { Location } from "history";
8  import { drawerClose } from "../ui";
9  import { stringify } from "./utilities";
10
11 export function* getLinkSaga(update: QueryString) {
12   let location: Location = yield select(getLocation);
13   let actualQueryString: QueryString = yield select(getQueryString);
14   return getLinkFromData(actualQueryString, location, update);
15 }
16
17 export function* pushQueryStringSaga(update: PayloadAction<QueryString>)
   ↪  {
18   let link: string = yield call(getLinkSaga, update.payload);
19   yield put(push(link));
20   yield put(drawerClose());
21 }
22
23 export function* closeDrawerSaga() {
24   yield put(drawerClose());
25 }
26
27 export default function* queryStringSaga() {
```

```
28    yield takeEvery(pushQueryString, pushQueryStringSaga);
29    yield takeEvery(CALL_HISTORY_METHOD, closeDrawerSaga);
30  }
```

Arquivo selectors.ts

```
1   import {
2     RouterRootState,
3     getSearch,
4     getLocation
5   } from "connected-react-router";
6   import { createSelector } from "reselect";
7   import { parse, stringify } from "./utilities";
8   import { QueryString } from "./types";
9   import { Location } from "history";
10
11  function getSearchString(state: RouterRootState): string {
12    return getSearch(state);
13  }
14
15  export function getLinkFromData(
16    actualQueryString: QueryString,
17    location: Location,
18    update: QueryString
19  ): string {
20    let newQueryString: QueryString = { ...actualQueryString, ...update };
21    let newSearch = stringify(newQueryString);
22    return location.pathname + newSearch;
23  }
24
25  export function getLinkSelector(update: QueryString) {
26    return function(state: RouterRootState): string {
27      let location: Location = getLocation(state);
28      let actualQueryString = getQueryString(state);
29      return getLinkFromData(actualQueryString, location, update);
30    };
31  }
32
33  export const getQueryString = createSelector(
34    getSearchString,
```

```
  parse
);
```

Arquivo types.ts

```
export type QueryString = { [id: string]: string | string[] };

export type URLQueryString = {
  url: string;
  querystring: QueryString;
};
```

Arquivo utilities.ts

```
import { QueryString } from "./types";

export function parse(search: string): QueryString {
  let result: QueryString = {};
  if (search.length < 2 || search[0] !== "?") {
    return result;
  }
  let parameters = search.substr(1).split("&");
  for (let parameter of parameters) {
    let [key, value] = parameter.split("=", 2);
    value = decodeURIComponent(value);
    let oldValue = result[key];
    if (Array.isArray(oldValue)) {
      oldValue.push(value);
    } else if (!oldValue) {
      result[key] = value;
    } else {
      result[key] = [oldValue, value];
    }
  }
  return result;
}

export function stringify(querystring: QueryString): string {
  let newSearchArray: string[] = [];
```

```
26    for (let [key, paramValue] of Object.entries(querystring)) {
27      let values = Array.isArray(paramValue) ? paramValue : [paramValue];
28      for (let value of values) {
29        newSearchArray.push(key + "=" + encodeURIComponent(value));
30      }
31    }
32    return newSearchArray.length > 0 ? "?" + newSearchArray.join("&") : "";
33  }
```

B.2.71 Pasta src/js/pages/base/store/ui

Arquivo index.ts

```
1  import { IModule } from "redux-dynamic-modules";
2  export * from "./slice";
3  import { reducer as UI } from "./slice";
4  import { FullState } from "./types";
5  export * from "./selectors";
6  export { FullState, SliceState } from "./types";
7
8  export default function getReduxModule(): IModule<FullState> {
9    return {
10     id: "ui",
11     reducerMap: {
12       UI
13     }
14   };
15 }
```

Arquivo reducers.ts

```
1  import { SliceState } from "./types";
2
3  export function drawerOpen(state: SliceState) {
4    state.drawerOpen = true;
5  }
6
7  export function drawerClose(state: SliceState) {
8    state.drawerOpen = false;
9  }
```

```
10
11 export function drawerToggle(state: SliceState) {
12   state.drawerOpen = !state.drawerOpen;
13 }
```

Arquivo selectors.ts

```
1 import { FullState } from "./types";
2
3 export function isDrawerOpen(state: FullState): boolean {
4   return state.UI.drawerOpen;
5 }
```

Arquivo slice.ts

```
1 import { createSlice } from "@reduxjs/toolkit";
2 import { SliceState } from "./types";
3 import * as reducers from "./reducers";
4
5 const uiSlice = createSlice({
6   name: "@@ui",
7   initialState: {
8     drawerOpen: false
9   } as SliceState,
10   reducers
11 });
12
13 const { actions, reducer } = uiSlice;
14
15 export const { drawerOpen, drawerClose, drawerToggle } = actions;
16 export { reducer };
```

Arquivo types.ts

```
1 export type SliceState = {
2   drawerOpen: boolean;
3 };
4
5 export type FullState = {
6   UI: SliceState;
7 };
```

B.2.72 Pasta src/js/pages/create-plan/store/period

Arquivo index.ts

```
import { createSlice, PayloadAction } from "@reduxjs/toolkit";
import { selectUniversity, ActionSelectUniversity } from "../university";
import { takeEvery, put } from "redux-saga/effects";
import gql from "graphql-tag";
import ApolloClient from "apollo-client";

export interface Period {
  id: string;
  name: string;
}

export type ActionPeriodsLoaded = PayloadAction<Period[]>;

export type ActionSelectPeriod = PayloadAction<Period>;

export interface State {
  periods: Period[];
  selected: Period | null;
}

const periodSlice = createSlice({
  name: "@@periods",
  initialState: {
    periods: [],
    selected: null
  } as State,
  reducers: {
    periodsLoaded: function(state, action: ActionPeriodsLoaded) {
      state.periods = action.payload;
    },
    selectPeriod: function(state, action: ActionSelectPeriod) {
      state.selected = action.payload;
    }
  },
  extraReducers: {
    [selectUniversity.toString()]: function(
```

```
37        state,
38        action: ActionSelectUniversity
39      ) {
40        state.periods = [];
41        state.selected = null;
42      }
43    }
44  });
45
46  const { actions, reducer } = periodSlice;
47  export const { periodsLoaded, selectPeriod } = actions;
48
49  export { reducer };
50
51  function* loadPeriods<CacheShape>(
52    client: ApolloClient<CacheShape>,
53    action: ActionSelectUniversity
54  ) {
55    const query = gql`
56      query($universityID: String!) {
57        periods(universityID: "$universityID") {
58          id
59          name
60        }
61      }
62    `;
63    let variables = {
64      universityID: action.payload.id
65    };
66    let periods = yield client.query<Period[]>({ query, variables });
67    put(periodsLoaded(periods.data));
68  }
69
70  export default function* saga<CacheShape>(client:
    ↪ ApolloClient<CacheShape>) {
71    yield takeEvery(selectUniversity.toString(), loadPeriods, client);
72  }
```

B.2.73 Pasta src/js/pages/create-plan/store/university

Arquivo index.ts

```
import { createSlice, PayloadAction, Action } from "@reduxjs/toolkit";
import Client from "../../../../client";
import gql from "graphql-tag";
import { put, takeEvery } from "redux-saga/effects";

export interface University {
  id: string;
  acronym: string;
  name: string;
}

export type ActionLoadUniversities = Action<"@@universities/LOAD">;

export type ActionUniversitiesLoaded = PayloadAction<University[]>;

export type ActionSelectUniversity = PayloadAction<University>;

const universitiesSlice = createSlice({
  name: "@@universities",
  initialState: {
    universities: [] as University[],
    selected: null as University | null
  },
  reducers: {
    loadUniversity: function(state, action) {
      state.selected = null;
    },
    universitiesLoaded: function(state, action: ActionUniversitiesLoaded)
    ↪ {
      state.universities = action.payload;
    },
    selectUniversity: function(state, action: ActionSelectUniversity) {
      state.selected = action.payload;
    }
  }
});
```

```
36
37 const { actions, reducer } = universitiesSlice;
38 // Extract and export each action creator by name
39 export const { loadUniversity, universitiesLoaded, selectUniversity } =
   ↪  actions;
40 // Export the reducer, either as a default or named export
41 export { reducer };
42
43 function* loadUniversities(action: ActionLoadUniversities) {
44   const query = gql`
45     query {
46       universities {
47         id
48         name
49       }
50     }
51   `;
52   let universities = yield Client.query(query);
53   put(universitiesLoaded(universities.data));
54 }
55
56 export function* saga() {
57   yield takeEvery(loadUniversity.toString(), loadUniversities);
58 }
```

B.2.74 Pasta src/js/pages/plan-details/components/cards

Arquivo discipline-offer.tsx

```
1  import React from "react";
2  import DisciplineOfferCard, {
3    DisciplineOfferCardModel
4  } from "../../../base/components/cards/discipline-offer";
5  import { useSelector, useDispatch } from "react-redux";
6  import { useDisciplineOfferLink } from "../link";
7  import {
8    getPossibilityDisciplineOfferColors,
9    getPossibilityDisciplineOffers,
10   loadDisciplineOffer,
11   disableDisciplineOffer,
```

```
12    enableDisciplineOffer,
13    removeDisciplineOffer
14  } from "../../store/plan";
15
16  const defaultSelection = {
17    enabled: false,
18    selected: false
19  };
20
21  type Props = {
22    item: DisciplineOfferCardModel;
23  };
24
25  export default function PlanDisciplineOfferCard({ item }: Props) {
26    let getDisciplineOfferLink = useDisciplineOfferLink();
27    let offers = useSelector(getPossibilityDisciplineOffers);
28    let colors =
      ↪  Object.values(useSelector(getPossibilityDisciplineOfferColors));
29    let dispatch = useDispatch();
30    return (
31      <DisciplineOfferCard
32        onDetails={getDisciplineOfferLink(item.id)}
33        disciplineOffer={item}
34        colors={colors}
35        onAction={() =>
36          dispatch(
37            !offers[item.id]
38              ? loadDisciplineOffer(item.id)
39              : offers[item.id].selection.enabled
40              ? disableDisciplineOffer(item.id)
41              : enableDisciplineOffer(item.id)
42          )
43        }
44        onDelete={
45          offers[item.id]
46            ? () => dispatch(removeDisciplineOffer(item.id))
47            : undefined
48        }
49        {...(offers[item.id]?.selection ?? defaultSelection)}
```

```
    />
  );
}
```

Arquivo discipline.tsx

```
import React from "react";
import DisciplineCard, {
  DisciplineCardModel
} from "../../../base/components/cards/discipline";
import { useSelector, useDispatch } from "react-redux";
import { useDisciplineLink } from "../link";
import {
  getPossibilityDisciplines,
  loadDiscipline,
  disableDiscipline,
  enableDiscipline,
  removeDiscipline
} from "../../store/plan";

const defaultSelection = {
  enabled: false
};

type Props = {
  item: DisciplineCardModel;
};

export default function PlanDisciplineCard({ item }: Props) {
  let getDisciplineLink = useDisciplineLink();
  let disciplines = useSelector(getPossibilityDisciplines);
  let dispatch = useDispatch();
  return (
    <DisciplineCard
      onDetails={getDisciplineLink(item.id)}
      discipline={item}
      {...(disciplines[item.id]?.selection ?? defaultSelection)}
      onAction={() =>
        dispatch(
          !disciplines[item.id]
```

```
35              ? loadDiscipline(item.id)
36              : disciplines[item.id].selection.enabled
37              ? disableDiscipline(item.id)
38              : enableDiscipline(item.id)
39          )
40        }
41        onDelete={
42          disciplines[item.id]
43            ? () => dispatch(removeDiscipline(item.id))
44            : undefined
45        }
46      />
47    );
48  }
```

Arquivo team.tsx

```
1  import React from "react";
2  import TeamCard, { TeamCardModel } from
   ↪  "../../../base/components/cards/team";
3  import { useSelector, useDispatch } from "react-redux";
4  import {
5    getPossibilityTeams,
6    getPossibilityDisciplineOfferColors,
7    loadDisciplineOffer,
8    disableTeam,
9    enableTeam,
10   removeTeam
11 } from "../../store/plan";
12 import { getLessUsedColor } from "../../store/plan/colors";
13 import { useTeamLink } from "../link";
14
15 const defaultSelection = {
16   enabled: false
17 };
18
19 type Props = {
20   item: TeamCardModel;
21 };
22
```

```
export default function PlanTeamCard({ item }: Props) {
  let getTeamLink = useTeamLink();
  let teams = useSelector(getPossibilityTeams);
  let colors = useSelector(getPossibilityDisciplineOfferColors);
  let dispatch = useDispatch();
  let color = getLessUsedColor(Object.values(colors));
  return (
    <TeamCard
      onDetails={getTeamLink(item.discipline.id, item.id)}
      team={item}
      color={colors[item.discipline.id] ?? color}
      onAction={() =>
        dispatch(
          !teams[item.id]
            ? loadDisciplineOffer(item.discipline.id)
            : teams[item.id].selection.enabled
            ? disableTeam({
                offerID: item.discipline.id,
                teamID: item.id
              })
            : enableTeam({ offerID: item.discipline.id, teamID: item.id
            ↪ })
        )
      }
      onDelete={
        teams[item.id] ? () => dispatch(removeTeam(item.id)) : undefined
      }
      {...(teams[item.id]?.selection ?? defaultSelection)}
    />
  );
}
```

B.2.75 Pasta src/js/pages/plan-details/components/tabs

Arquivo calendar-tabs.tsx

```
import React, { useCallback } from "react";
import { getPlanView, View, updateView } from "../../store/ui";
import { useSelector, useDispatch } from "react-redux";
import { TabBar, Tab, TabBarOnActivateEventT } from "@rmwc/tabs";
```

```
5
6  export default function CalendarTabs() {
7    let plan = useSelector(getPlanView);
8    let dispatch = useDispatch();
9    let tabs: View[] = ["calendar-week", "calendar-three-days",
     ↪  "calendar-day"];
10   let activate = useCallback(
11     (evt: TabBarOnActivateEventT) =>
12       dispatch(updateView({ view: tabs[evt.detail.index] })),
13     [dispatch]
14   );
15   return (
16     <TabBar activeTabIndex={tabs.indexOf(plan)} onActivate={activate}>
17       <Tab>Semana</Tab>
18       <Tab>Três dias</Tab>
19       <Tab>Dia</Tab>
20     </TabBar>
21   );
22 }
```

Arquivo list-tabs.tsx

```
1  import React, { useCallback } from "react";
2  import { getPlanView, View, updateView } from "../../store/ui";
3  import { useSelector, useDispatch } from "react-redux";
4  import { TabBar, Tab, TabBarOnActivateEventT } from "@rmwc/tabs";
5
6  export default function ListTabs() {
7    let plan = useSelector(getPlanView);
8    let dispatch = useDispatch();
9    let tabs: View[] = [
10     "list-teams",
11     "list-discipline-offers",
12     "list-disciplines"
13   ];
14   let activate = useCallback(
15     (evt: TabBarOnActivateEventT) =>
16       dispatch(updateView({ view: tabs[evt.detail.index] })),
17     [dispatch]
```

```
18    );
19    return (
20      <TabBar activeTabIndex={tabs.indexOf(plan)} onActivate={activate}>
21        <Tab>Turmas</Tab>
22        <Tab>Ofertas de Disciplinas</Tab>
23        <Tab>Disciplinas</Tab>
24      </TabBar>
25    );
26  }
```

Arquivo search-tabs.tsx

```
1   import React, { useCallback } from "react";
2   import { getPlanView, View, updateView } from "../../store/ui";
3   import { useSelector, useDispatch } from "react-redux";
4   import { TabBar, Tab, TabBarOnActivateEventT } from "@rmwc/tabs";
5
6   export default function SearchTabs() {
7     let plan = useSelector(getPlanView);
8     let dispatch = useDispatch();
9     let tabs: View[] = [
10      "search-teams",
11      "search-discipline-offers",
12      "search-disciplines",
13      "search-teachers"
14    ];
15    let activate = useCallback(
16      (evt: TabBarOnActivateEventT) =>
17        dispatch(updateView({ view: tabs[evt.detail.index] })),
18      [dispatch]
19    );
20    return (
21      <TabBar activeTabIndex={tabs.indexOf(plan)} onActivate={activate}>
22        <Tab>Turmas</Tab>
23        <Tab>Ofertas de Disciplinas</Tab>
24        <Tab>Disciplinas</Tab>
25        <Tab>Professores</Tab>
26      </TabBar>
```

```
  );
}
```

B.2.76 Pasta src/js/pages/plan-details/components/views

Arquivo calendar.tsx

```tsx
import React, { useCallback } from "react";
import {
  CalendarInteractiveProps,
  CalendarInteractive,
  Schedule
} from "../../../base/components/calendar";
import { useSelector, useDispatch } from "react-redux";
import {
  getSelectedCalendarSchedules,
  getPossibilityTeams
} from "../../store/plan";
import { useDisciplineOfferLink } from "../link";
import PlanContainer from "../container";
import CalendarTabs from "../tabs/calendar-tabs";
import { updateView } from "../../store/ui";

export default function Calendar(
  props: Pick<CalendarInteractiveProps, "visibleDays">
) {
  let schedules = useSelector(getSelectedCalendarSchedules);
  let teams = useSelector(getPossibilityTeams);
  let dispatch = useDispatch();
  let scheduleClick = useCallback(
    function(schedule: Schedule) {
      if (!schedule.id) {
        return;
      }
      let discipline = teams[schedule.id]?.team?.discipline;
      if (!discipline) {
        return;
      }
      dispatch(
        updateView({
```

```
            view: "team",
            offer_id: discipline.id,
            team_id: schedule.id
          })
        );
      },
      [dispatch, teams]
    );
    return (
      <PlanContainer>
        <CalendarTabs />
        <CalendarInteractive
          interval={30}
          hourHeight={45}
          onScheduleClick={scheduleClick}
          slide
          schedules={schedules}
          {...props}
        />
      </PlanContainer>
    );
}
```

Arquivo combinations.tsx

```
import React, { useEffect, useState, useCallback, Fragment } from
↪  "react";
import PlanContainer from "../container";
import { useSelector, useDispatch } from "react-redux";
import {
  getArrayCombinator,
  ArrayCombinatorWrapper
} from "../../store/combinations/helpers";
import { getStateOptions } from "../../store/combinations";
import {
  Team,
  getPossibilityDisciplineOfferColors,
  setSelectedTeams,
  getSelectedTeamsIndexesGroupedByDisciplines
```

```
} from "../../store/plan";
import {
  CombinatorResult,
  CombinatorIndex
} from "../../store/combinations/combinator/combinations";
import { Grid, GridCell } from "@rmwc/grid";
import InfiniteScroll, {
  InfiniteScrollRenderOptions
} from "../../../base/components/infinite-scroll";
import CombinationCard from "../../../base/components/cards/combination";
import { CalendarProvider } from "../../../base/components/calendar";
import { LinearProgress } from "@rmwc/linear-progress";

type CombinationsRenderProps = InfiniteScrollRenderOptions<
  CombinatorResult<Team>
> & {
  offerColors: { [id: string]: string };
};

function Loading({
  loading,
  itemCount
}: Pick<CombinationsRenderProps, "loading" | "itemCount">) {
  if (!loading) {
    return (
      <GridCell span={12} key="label">
        {itemCount} combinações encontradas:
      </GridCell>
    );
  }
  return (
    <Fragment>
      <GridCell span={12}>
        <LinearProgress />
      </GridCell>
      <GridCell span={12}>Computando combinações...</GridCell>
    </Fragment>
  );
}
```

```
53
54 function checkEqual(a: CombinatorIndex[], b: CombinatorIndex[]): boolean
   ↪ {
55   let match = a.length === b.length;
56   for (let i = 0; match && i < a.length; i++) {
57     match =
58       a[i].mainIndex === b[i].mainIndex && a[i].subIndex ===
         ↪  b[i].subIndex;
59   }
60   return match;
61 }
62
63 function CombinationsRender(props: CombinationsRenderProps) {
64   let { loading, offerColors, start, itemCount, items } = props;
65   let dispatch = useDispatch();
66   let selected =
     ↪  useSelector(getSelectedTeamsIndexesGroupedByDisciplines);
67   return (
68     <Grid>
69       <Loading loading={loading} itemCount={itemCount} />
70       {items.map((item: CombinatorResult<Team>, i) => (
71         <GridCell desktop={4} tablet={4} phone={4} key={i + start + 1}>
72           <CombinationCard
73             teams={item.results}
74             selected={checkEqual(item.indexes, selected)}
75             index={i + start + 1}
76             offerColors={offerColors}
77             onSelect={() => dispatch(setSelectedTeams(item.results))}
78           />
79         </GridCell>
80       ))}
81     </Grid>
82   );
83 }
84
85 type combinationsResult = {
86   results: CombinatorResult<Team>[];
87   count: number | null;
88   loading: boolean;
```

```
89    error: boolean;
90  };
91
92  export default function Combinations() {
93    let offerColors = useSelector(getPossibilityDisciplineOfferColors);
94    let stateOptions = useSelector(getStateOptions);
95    let [result, setResult] = useState<combinationsResult>({
96      error: false,
97      count: null,
98      loading: true,
99      results: []
100   });
101   let fetchMoreCallback = useCallback(
102     function() {
103       setResult({
104         ...result,
105         loading: true
106       });
107       let combinatorPromise = getArrayCombinator(stateOptions,
      ↪  result.count);
108       combinatorPromise.then(
109         async function(combinator: ArrayCombinatorWrapper<Team>) {
110           let rows: CombinatorResult<Team>[] = result.results.slice();
111           let start = result.results.length;
112           for (let c = 0; c < 10 && c + start < combinator.length; c++) {
113             let row = await combinator.getIndex(c + start);
114             rows.push(row);
115           }
116           combinator.dispose();
117           setResult({
118             error: false,
119             loading: false,
120             count: combinator.length,
121             results: rows
122           });
123         },
124         function() {
125           setResult({
126             error: true,
```

```
            loading: false,
            results: [],
            count: null
          });
        }
      );
    },
    [stateOptions, result]
  );
  useEffect(
    function() {
      fetchMoreCallback();
    },
    [stateOptions]
  );
  if (result.error) {
    return (
      <PlanContainer>
        <div className="results">Erro ao gerar as combinações.</div>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <div className="results">
        <CalendarProvider width={344} height={194}>
          <InfiniteScroll
            itemCount={result.count ?? 0}
            loading={result.loading}
            loadMore={fetchMoreCallback}
            items={result.results}
            render={CombinationsRender}
            renderOptions={{ offerColors }}
          />
        </CalendarProvider>
      </div>
    </PlanContainer>
  );
}
```

Arquivo settings.tsx

```tsx
import React from "react";
import {
  useTextField,
  useSubmit,
  useCheckboxField
} from "../../../base/components/form";
import { Button } from "@rmwc/button";
import { GridCell, Grid, GridInner } from "@rmwc/grid";
import { TextField } from "@rmwc/textfield";
import { Snackbar } from "@rmwc/snackbar";
import { getPlanFull, setPlanSettings } from "../../store/plan";
import { useSelector, useDispatch } from "react-redux";
import PlanContainer from "../container";
import { Switch } from "@rmwc/switch";
import { Dispatch } from "redux";

type Fields = {
  name: string;
  isPublic: boolean;
};

async function submit(dispatch: Dispatch, fields: Fields) {
  dispatch(
    setPlanSettings({
      name: fields.name,
      public: fields.isPublic
    })
  );
  return Promise.resolve(1);
}

export default function Settings() {
  let plan = useSelector(getPlanFull);
  let [name, nameField] = useTextField(plan?.name);
  let [isPublic, publicField] = useCheckboxField(plan?.public);
  let dispatch = useDispatch();
```

```
37    let { submitted, error, clearError, onSubmit } = useSubmit((evt,
      ↪  csrfToken) =>
38      submit(dispatch, { name, isPublic })
39    );
40    if (submitted) {
41      return (
42        <PlanContainer>
43          <p>Plano modificado com sucesso!</p>
44        </PlanContainer>
45      );
46    }
47    return (
48      <PlanContainer>
49        <form onSubmit={onSubmit}>
50          <Grid>
51            <GridCell span={12} className="form-row">
52              <TextField
53                name="name"
54                type="text"
55                label="Nome do Plano"
56                {...nameField}
57              />
58            </GridCell>
59            <GridCell span={12} className="form-row">
60              <Switch name="public" label="Publica" {...publicField} />
61            </GridCell>
62            <GridCell span={12} className="form-row">
63              <Button type="submit" raised>
64                Editar
65              </Button>
66            </GridCell>
67          </Grid>
68          {error ? <Snackbar open onClose={clearError} message={error} /> :
            ↪  null}
69        </form>
70      </PlanContainer>
71    );
72  }
```

Arquivo versions.tsx

```
import React from "react";
import {
  List,
  ListItem,
  ListItemText,
  ListItemPrimaryText,
  ListItemSecondaryText
} from "@rmwc/list";
import { useDispatch, useSelector } from "react-redux";
import {
  selectVersion,
  getPlanID,
  getSelectedVersionIndex
} from "../../store/plan";
import PlanContainer from "../container";
import { useQuery } from "@apollo/react-hooks";
import { loadPlanVersions } from "../../queries/load-plan.gql";
import {
  RemoteLoadPlanVersionsQuery,
  RemoteLoadPlanVersionsQueryVariables
} from "../../../../graphql";
import { LinearProgress } from "@rmwc/linear-progress";
import { Grid, GridCell } from "@rmwc/grid";
import { PlanCard } from "../../../base/components/cards/plan";
import { PlanVersionCard } from
↪  "../../../base/components/cards/plan-version";
import { CalendarProvider } from "../../../base/components/calendar";

export default function Versions() {
  let dispatch = useDispatch();
  let versionID = useSelector(getSelectedVersionIndex);
  let planID = useSelector(getPlanID);
  if (!planID) {
    return (
      <PlanContainer>
        <p>Requisição inválida: Parâmetros inválidos foram informados</p>
      </PlanContainer>
```

```
    );
  }
  let { data, loading, error } = useQuery<
    RemoteLoadPlanVersionsQuery,
    RemoteLoadPlanVersionsQueryVariables
  >(loadPlanVersions, {
    variables: {
      planID
    }
  });
  if (error) {
    return (
      <PlanContainer>
        <p>Erro ao fazer o carregamento dos dados de versão</p>
      </PlanContainer>
    );
  }
  if (loading) {
    return (
      <PlanContainer>
        <LinearProgress />
        <p>Carregando dados..</p>
      </PlanContainer>
    );
  }
  if (!data || !data.plan) {
    return (
      <PlanContainer>
        <p>Plano não encontrado.</p>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <CalendarProvider width={344} height={194}>
        <div className="versions">
          <Grid>
            {data.plan.versions.map(version => (
```

```
75                <GridCell desktop={4} phone={4} tablet={4}
                  ↪  key={version.id}>
76                  <PlanVersionCard
77                    selected={versionID === version.id}
78                    version={version}
79                    onAction={() => dispatch(selectVersion(version.id))}
80                  />
81                </GridCell>
82              ))}
83              {data.plan.versions.length === 0 ? (
84                <GridCell span={12}>
85                  Isso não deveria acontecer mas...nenhuma versão foi
                    ↪  encontrada
86                </GridCell>
87              ) : null}
88            </Grid>
89          </div>
90        </CalendarProvider>
91      </PlanContainer>
92    );
93  }
```

B.2.77 Pasta src/js/pages/plan-details/store/combinations

Arquivo actions.ts

```
1  import { createAction } from "@reduxjs/toolkit";
```

Arquivo helpers.ts

```
1  import manager from "./manager";
2  import { CombinatorStateOptions } from "./types";
3  import { ArrayCombinator } from "./combinator/combinations";
4  import { CombinatorHandler } from "./combinator/client";
5  import { Team } from "../plan";
6
7  export async function getCombinator(
8    stateOptions: CombinatorStateOptions
9  ): Promise<CombinatorHandler<Team>> {
10   let { options, data, selected } = stateOptions;
```

```
  let combinator = await manager.getCombinator(data, options);
  if (selected.length > 0 && combinator) {
    await combinator.set(selected);
  }
  return combinator;
}

export class ArrayCombinatorWrapper<T> extends ArrayCombinator<T> {
  protected _disposer: CombinatorHandler<T>;
  constructor(combinator: CombinatorHandler<T>, length: number) {
    super(combinator, length);
    this._disposer = combinator;
  }
  dispose(): Promise<void> {
    return this._disposer.dispose();
  }
}

export async function getArrayCombinator(
  stateOptions: CombinatorStateOptions,
  length: number | null = null
): Promise<ArrayCombinatorWrapper<Team>> {
  let combinator = await getCombinator(stateOptions);
  if (length !== undefined) {
    length = await combinator.count();
  }
  return new ArrayCombinatorWrapper(combinator, length);
}
```

Arquivo index.ts

```
export * from "./slice";
export * from "./actions";
export * from "./helpers";
export * from "./selectors";
import { reducer as combinations } from "./slice";
import { FullState } from "./types";
export * from "./types";
import { IModule } from "redux-dynamic-modules";

```

```
export default function getReduxModule(): IModule<FullState> {
  return {
    id: "plan-combinations",
    reducerMap: {
      combinations
    }
  };
}
```

Arquivo manager.ts

```
    import Combinator from "./combinator/client";

export default new Combinator();
```

Arquivo reducers.ts

```
import { SliceState, Limits } from "./types";
import { PayloadAction } from "@reduxjs/toolkit";

export function setClassLimits(
  { options }: SliceState,
  { payload }: PayloadAction<Limits>
) {
  options.minClass = payload.minClass;
  options.maxClass = payload.maxClass;
  options.minutesPerClass = payload.minutesPerClass;
}

export function enableDay(
  { options }: SliceState,
  { payload }: PayloadAction<number>
) {
  options.days = { ...options.days, [payload]: true };
}

export function disableDay(
  { options }: SliceState,
  { payload }: PayloadAction<number>
```

```
23 ) {
24   options.days = { ...options.days, [payload]: false };
25 }
26
27 export function enableHour(
28   { options }: SliceState,
29   { payload }: PayloadAction<string>
30 ) {
31   options.hours = { ...options.hours, [payload]: true };
32 }
33
34 export function disableHour(
35   { options }: SliceState,
36   { payload }: PayloadAction<string>
37 ) {
38   options.hours = { ...options.hours, [payload]: false };
39 }
40
41 export function enableTeacher(
42   { options }: SliceState,
43   { payload }: PayloadAction<string>
44 ) {
45   options.teachers = { ...options.teachers, [payload]: true };
46 }
47
48 export function disableTeacher(
49   { options }: SliceState,
50   { payload }: PayloadAction<string>
51 ) {
52   options.teachers = { ...options.teachers, [payload]: false };
53 }
54
55 export function enableDiscipline(
56   { options }: SliceState,
57   { payload }: PayloadAction<string>
58 ) {
59   options.disciplines = { ...options.disciplines, [payload]: true };
60 }
61
```

```
export function enableTeam(
  { options }: SliceState,
  { payload }: PayloadAction<string>
) {
  options.teams = { ...options.teams, [payload]: true };
}

export function disableDiscipline(
  { options }: SliceState,
  { payload }: PayloadAction<string>
) {
  options.disciplines = {
    ...options.disciplines,
    [payload]: false
  };
}

export function disableTeam(
  { options }: SliceState,
  { payload }: PayloadAction<string>
) {
  options.teams = { ...options.teams, [payload]: false };
}
```

Arquivo sagas.ts

```
import { Team, setSelectedTeams } from "../plan";
import { CombinatorHandler } from "./combinator/client";
import { select, take, call, put, fork } from "redux-saga/effects";
import { getCombinator } from "./helpers";
import { getStateOptions } from "./selectors";
import { CombinatorStateOptions } from "./types";
import { CombinatorResult } from "./combinator/combinations";

export function* getCombinatorSaga() {
  let options: CombinatorStateOptions = yield select(getStateOptions);
  let combinator: CombinatorHandler<Team> = yield call(getCombinator,
  ↪  options);
  return combinator;
}
```

```
14
15 function* findNextCombinations(options: CombinatorStateOptions) {
16   let match = true;
17   let combinator: CombinatorHandler<Team> = yield call(getCombinator,
     ↪  options);
18   try {
19     let result: CombinatorResult<Team> = yield call([
20       combinator,
21       combinator.next
22     ]);
23     yield put(setSelectedTeams(result.results));
24   } catch (e) {
25     match = false;
26   } finally {
27     yield call([combinator, combinator.dispose]);
28   }
29   return match;
30 }
31
32 export function* findNewCombination() {
33   let options: CombinatorStateOptions = yield select(getStateOptions);
34   if (options.data.length > options.options.max) {
35     return false;
36   }
37   let data: Team[][] = options.data.slice();
38   for (let { mainIndex, subIndex } of options.selected) {
39     data[mainIndex] = [data[mainIndex][subIndex]];
40   }
41   let match = yield call(findNextCombinations, {
42     ...options,
43     data,
44     selected: []
45   });
46   if (!match) {
47     match = yield call(findNextCombinations, options);
48   }
49   return match;
50 }
```

Arquivo selectors.ts

```
import {
  FullState,
  SliceState,
  Options,
  CombinatorStateOptions
} from "./types";
import {
  FullState as PlanFullState,
  getEnabledDisciplineOffers,
  getEnabledTeamsGroupedByDisciplines,
  getSelectedTeamsIndexesGroupedByDisciplines
} from "../plan";
import { RouterRootState } from "connected-react-router";

export function getCombinationsSlice(state: FullState): SliceState {
  return state.combinations;
}

export function getOptions(state: FullState): Options {
  return getCombinationsSlice(state).options;
}

export function getMax(state: FullState): number {
  let { max } = getCombinationsSlice(state);
  return max === undefined ? Infinity : max;
}

export function getMin(state: FullState): number {
  let { min } = getCombinationsSlice(state);
  return min === undefined ? 1 : min;
}

function respectInterval(value: number, count: number) {
  let result = Math.max(value, 1);
  result = Math.min(result, count);
  return result;
}
```

```
38
39 export function getStateOptions(
40   state: FullState & PlanFullState & RouterRootState
41 ): CombinatorStateOptions {
42   let ruleOptions = getOptions(state);
43   let max = getMax(state);
44   let min = getMin(state);
45   let { length } = Object.keys(getEnabledDisciplineOffers(state));
46   min = respectInterval(min, length);
47   max = respectInterval(max, length);
48   let data = getEnabledTeamsGroupedByDisciplines(state);
49   let selected = getSelectedTeamsIndexesGroupedByDisciplines(state);
50   return {
51     data,
52     options: {
53       min: Math.min(min, max),
54       max: Math.max(min, max),
55       options: ruleOptions
56     },
57     selected
58   };
59 }
```

Arquivo slice.ts

```
1 import { createSlice } from "@reduxjs/toolkit";
2 import * as reducers from "./reducers";
3 import { SliceState } from "./types";
4
5 const initialState: SliceState = {
6   options: {}
7 };
8
9 let { actions, reducer } = createSlice({
10   initialState,
11   name: "@@combinations",
12   reducers
13 });
14
15 export const {
```

```
  setClassLimits,
  enableDay,
  disableDay,
  enableHour,
  disableHour,
  enableTeacher,
  disableTeacher,
  enableDiscipline,
  disableDiscipline,
  enableTeam,
  disableTeam
} = actions;
export { reducer };
```

Arquivo types.ts

```
import { CombinationOptions } from "./combinator/types";
import { CombinatorIndex } from "./combinator/combinations";
import { Team } from "../plan";

export type Limits = {
  minClass?: number;
  maxClass?: number;
  minutesPerClass?: number;
};

export type Options = Limits & {
  days?: { [day: number]: boolean };
  hours?: { [hour: number]: boolean };
  teachers?: { [teacher: string]: boolean };
  disciplines?: { [discipline: string]: boolean };
  teams?: { [team: string]: boolean };
};

export type SliceState = {
  options: Options;
  min?: number;
  max?: number;
};

```

```
25 export type FullState = {
26   combinations: SliceState;
27 };
28
29 export type CombinatorStateOptions = {
30   data: Team[][];
31   options: CombinationOptions;
32   selected: CombinatorIndex[];
33 };
```

B.2.78 Pasta src/js/pages/plan-details/store/plan

Arquivo actions.ts

```
1  import { createAction, PayloadAction } from "@reduxjs/toolkit";
2  import { PayloadPlanID, AddVersion } from "./types";
3  import { pushQueryString } from "../../../base/store/querystring";
4
5  export const loadDiscipline =
   ↪  createAction<string>("@@plan/LOAD_DISCIPLINE");
6  export const loadDisciplineOffer = createAction<string>(
7    "@@plan/LOAD_DISCIPLINE_OFFER"
8  );
9  export const loadPlan = createAction<PayloadPlanID>("@@plan/LOAD_PLAN");
10 export function selectVersion(version: string) {
11   return pushQueryString({
12     version
13   });
14 }
15
16 export const addVersion = createAction<AddVersion<any>>(
17   "@@plan/internal/ADD_VERSION"
18 );
```

Arquivo colors.ts

```
1  import { PlanPossibility } from "./types";
2
3  // Source:
   ↪  https://gka.github.io/palettes/#/16|d|7587bf,96ffea,cad792|ffffe0,a1a1a1,a26b67|1|1
```

```
let pallete = [
  "#7587bf",
  "#7992c3",
  "#7d9cc6",
  "#82a7c9",
  "#87b1cc",
  "#8dbccd",
  "#94c6ce",
  "#9cd1cc",
  "#eeedd4",
  "#dedbc7",
  "#cfc9ba",
  "#c2b7ac",
  "#b7a59d",
  "#ad938d",
  "#a6807b",
  "#a26b67"
];

type CountAggregation = {
  color: string;
  count: number;
};

type Count = {
  old: CountAggregation;
  actual: CountAggregation;
};

function getActual(color: string): CountAggregation {
  return { color, count: 0 };
}

function aggregate(result: Count, color: string) {
  if (result.actual.color !== color) {
    if (result.actual.count < result.old.count) {
      result.old = { ...result.actual };
    }
    result.actual = getActual(color);
```

```
43    }
44    result.actual.count++;
45    return result;
46  }
47
48  function isValidColor(color: string) {
49    return pallete.includes(color);
50  }
51
52  export function getLessUsedColor(colors: string[]): string {
53    const color =
54      pallete.find(function(actualColor) {
55        return !colors.includes(actualColor);
56      }) || pallete[0];
57    const count = colors.length + 1;
58    const { old, actual } = colors
59      .filter(isValidColor)
60      .sort()
61      .reduce(aggregate, {
62        actual: getActual(color),
63        old: {
64          color,
65          count
66        }
67      });
68    let lessPopular =
69      old.count !== count && actual.count < old.count ? actual : old;
70    return lessPopular.color;
71  }
72
73  function getColorsUsed({ selectionTeams, offers }: PlanPossibility):
    ↪  string[] {
74    let colors: string[] = [];
75    let discipline = new Set<string>();
76    for (let { offerID } of selectionTeams) {
77      if (!discipline.has(offerID)) {
78        let { color } = offers[offerID];
79        if (color.length > 0) {
80          discipline.add(offerID);
```

```
        colors.push(color);
      }
    }
  }
  return colors;
}

export default function getColor(possibility: PlanPossibility): string {
  const colors = getColorsUsed(possibility);
  return getLessUsedColor(colors);
}
```

Arquivo index.ts

```
export * from "./slice";
export * from "./actions";
export * from "./selectors";
import { reducer as plan } from "./slice";
import { FullState } from "./types";
import saga from "./sagas";
export * from "./types";
import Optimist from "redux-optimist";
import { ISagaModule } from "redux-dynamic-modules-saga";
import client from "../../../../client";

export default function getReduxModule(): ISagaModule<FullState> {
  return {
    id: "plan",
    reducerMap: {
      plan: Optimist(plan)
    },
    sagas: [
      {
        saga,
        argument: client
      }
    ]
  };
}
```

Arquivo loader.ts

```
import {
  RemoteLoadPlanVersionQuery,
  RemoteGetPlanVersionDataFragment
} from "../../../../graphql";
import { PlanVersion, PlanPossibility, SliceState } from "./types";
import getColor from "./colors";

interface UpdatedVersion {
  version: string;
}

function getUpdatedVersion(
  updated: UpdatedVersion | undefined
): string | undefined {
  return updated ? updated.version : updated;
}

export function loadVersion(
  result: SliceState,
  version: RemoteGetPlanVersionDataFragment
): PlanVersion {
  let resultVersion: PlanVersion = {
    id: version.id,
    savedAt: version.savedAt,
    defaultPossibility: version.selectedPossibility,
    possibilities: []
  };
  for (let possibility of version.possibilities) {
    let resultPossibility: PlanPossibility = {
      name: possibility.name,
      disciplines: {},
      offers: {},
      selectionDisciplines: [],
      selectionTeams: []
    };
    for (let discipline of possibility.disciplines) {
      resultPossibility.disciplines[discipline.disciplineID] =
```

```
38          discipline.disciplineVersion;
39        resultPossibility.selectionDisciplines.push(discipline);
40      }
41      let emptyColor = false;
42      for (let team of possibility.teams) {
43        let { color } = team;
44        emptyColor = emptyColor || color.length === 0;
45        resultPossibility.offers[team.offerID] = {
46          version: team.offerVersion,
47          color
48        };
49        resultPossibility.selectionTeams.push(team);
50      }
51      if (emptyColor) {
52        for (let { offerID } of possibility.teams) {
53          let offer = resultPossibility.offers[offerID];
54          if (offer.color.length === 0) {
55            offer.color = getColor(resultPossibility);
56          }
57          resultPossibility.offers[offerID] = offer;
58        }
59      }
60      resultVersion.possibilities.push(resultPossibility);
61    }
62    for (let { deleted, discipline, updated } of version.disciplines) {
63      if (!result.disciplines[discipline.id]) {
64        result.disciplines[discipline.id] = {};
65      }
66      result.disciplines[discipline.id][discipline.version] = {
67        deleted,
68        discipline,
69        updatedVersion: getUpdatedVersion(updated)
70      };
71    }
72    for (let { custom, deleted, offer, updated } of
      ↪  version.disciplineOffers) {
73      let basicDiscipline = {
74        ...offer,
75        teamVersions: {}
```

```
    };
    let basicOffer = {
      deleted,
      updatedVersion: getUpdatedVersion(updated)
    };
    if (!result.offers[offer.id]) {
      result.offers[offer.id] = {};
    }
    if (custom) {
      result.offers[offer.id][offer.version] = {
        ...basicOffer,
        discipline: { ...basicDiscipline, custom },
        custom
      };
      for (let team of offer.teams) {
        if (!result.teams[team.id]) {
          result.teams[team.id] = {};
        }
        result.teams[team.id][team.version] = {
          ...team,
          discipline: offer,
          custom
        };
      }
    } else {
      result.offers[offer.id][offer.version] = {
        ...basicOffer,
        discipline: { ...basicDiscipline, custom },
        custom
      };
      for (let team of offer.teams) {
        if (!result.teams[team.id]) {
          result.teams[team.id] = {};
        }
        result.teams[team.id][team.version] = {
          ...team,
          discipline: offer,
          custom
        };
```

```
115       }
116     }
117     for (let team of offer.teams) {
118
          ↪  result.offers[offer.id][offer.version].discipline.teamVersions[team.id]
          ↪  =
119           team.version;
120     }
121   }
122   return resultVersion;
123 }
124
125 export function loadPlan(
126   result: SliceState,
127   { plan }: RemoteLoadPlanVersionQuery
128 ) {
129   if (!plan?.loadedVersion) {
130     return;
131   }
132   result.offers = result.offers || {};
133   result.disciplines = result.disciplines || {};
134   result.teams = result.teams || {};
135   result.plan = {
136     ...plan,
137     loadedVersion: loadVersion(result, plan.loadedVersion),
138     drafts: {},
139     draftsOrder: []
140   };
141 }
```

Arquivo sagas.ts

```
1 import { PayloadAction } from "@reduxjs/toolkit";
2 import { MetaVersion, BaseVersionPossibility, AddVersion } from
  ↪  "./types";
3 import { takeEvery, put, select, call, take } from "redux-saga/effects";
4 import {
5   pushQueryString,
6   getQueryString,
7   QueryString
```

```
} from "../../../base/store/querystring";
import {
  addVersion,
  selectVersion,
  loadDiscipline as loadDisciplineAction,
  loadDisciplineOffer as loadDisciplineOfferAction
} from "./actions";
import {
  getBaseVersionPossibility,
  getRemotePlanInput,
  getPlanID,
  getPlanLoadedVersionID
} from "./selectors";
import ApolloClient, { ApolloQueryResult } from "apollo-client";
import {
  loadPlanVersion,
  loadLatestPlanVersionID
} from "../../queries/load-plan.gql";
import {
  loadDiscipline,
  loadDisciplineOffer
} from "../../queries/search-load.gql";
import { savePlan } from "../../queries/save-plan.gql";
import {
  RemoteLoadPlanVersionQuery,
  RemoteSavePlanMutation,
  RemoteLoadLatestPlanVersionIdQuery,
  RemoteLoadDisciplineQuery,
  RemoteLoadDisciplineOfferQuery
} from "../../../../graphql";
import {
  resetPlan,
  errorPlan,
  loadingPlan,
  loadedDiscipline,
  loadedDisciplineOffer,
  loadedPlan,
  setSelectedTeams
} from "./slice";
```

```
import { LOCATION_CHANGE } from "connected-react-router";
import { FetchResult } from "apollo-link";
import { findNewCombination } from "../combinations/sagas";

export function* addVersionSaga<Shape>(
  client: ApolloClient<Shape>,
  action: PayloadAction<AddVersion<PayloadAction<any>>>
) {
  let baseVersionPossibility: BaseVersionPossibility = yield select(
    getBaseVersionPossibility
  );
  let { targetVersion, originalPayload } = action.payload;
  let newAction: MetaVersion<any> = {
    ...originalPayload,
    meta: {
      targetVersion,
      ...baseVersionPossibility
    }
  };
  yield put(newAction);
  if (originalPayload.type !== setSelectedTeams.toString()) {
    let match = yield call(findNewCombination);
    if (match) {
      return;
    }
  }
  let plan = yield select(getRemotePlanInput);
  try {
    let result: FetchResult<RemoteSavePlanMutation> = yield call(
      [client, client.mutate],
      {
        mutation: savePlan,
        variables: {
          plan
        }
      }
    );
    if (!result.data?.setPlan?.version) {
      throw new Error();
```

```
86      }
87      yield put(selectVersion(result.data.setPlan.version.id));
88    } catch (e) {
89      yield put(errorPlan("Erro durante o salvamento do plano"));
90    }
91  }
92
93  export function* loadPlanSaga<Shape>(client: ApolloClient<Shape>) {
94    yield put(loadingPlan());
95    let result: ApolloQueryResult<RemoteLoadPlanVersionQuery>;
96    let data: QueryString = yield select(getQueryString);
97    let planId: string | null = yield select(getPlanID);
98    let planLoadedVersion: string | null = yield
      ↪  select(getPlanLoadedVersionID);
99    let id = data.id;
100   if (Array.isArray(id)) {
101     id = id[0];
102   }
103   let message = "Erro durante o carregamento do plano";
104   if (typeof id === "string") {
105     let version = data.version;
106     if (Array.isArray(version)) {
107       version = version[0];
108     }
109     if (typeof version !== "string") {
110       let versionResult:
          ↪  ApolloQueryResult<RemoteLoadLatestPlanVersionIdQuery>;
111       versionResult = yield call([client, client.query], {
112         query: loadLatestPlanVersionID,
113         variables: {
114           planID: id
115         }
116       });
117       if (!versionResult.errors) {
118         version = versionResult.data?.plan?.version?.id ?? "";
119       }
120       if (!!version) {
121         yield put(selectVersion(version));
122       }
```

```
123     }
124     if (version != "" && planId === id && planLoadedVersion === version)
        ↪ {
125       yield put(loadedPlan());
126       return;
127     }
128     if (typeof version === "string" && version !== "") {
129       try {
130         result = yield call([client, client.query], {
131           query: loadPlanVersion,
132           variables: {
133             planID: id,
134             planVersion: version
135           }
136         });
137         if (!result.errors && result.data) {
138           yield put(resetPlan(result.data));
139           return;
140         }
141       } catch (e) {
142         message =
143           "Houve um erro durante o carregametno do plano. Verifique o ID
              ↪  e versão e tente novamente.";
144       }
145     }
146   } else {
147     message = "Requisição inválida: Configure um ID de plano para
        ↪  carregar";
148   }
149   yield put(errorPlan(message));
150 }
151
152 function* loadDisciplineSaga<Shape>(
153   client: ApolloClient<Shape>,
154   { payload: disciplineID }: PayloadAction<string>
155 ) {
156   try {
157     let result: ApolloQueryResult<RemoteLoadDisciplineQuery>;
158     result = yield call([client, client.query], {
```

```
159       query: loadDiscipline,
160       variables: {
161         disciplineID
162       }
163     });
164     if (!!result.errors || !result.data.discipline) {
165       throw new Error();
166     }
167     yield put(loadedDiscipline(result.data.discipline));
168   } catch (e) {
169     yield put(errorPlan("Erro durante o carregamento da disciplina"));
170   }
171 }
172
173 function* loadDisciplineOfferSaga<Shape>(
174   client: ApolloClient<Shape>,
175   { payload: offerID }: PayloadAction<string>
176 ) {
177   let result: ApolloQueryResult<RemoteLoadDisciplineOfferQuery>;
178   result = yield call([client, client.query], {
179     query: loadDisciplineOffer,
180     variables: {
181       offerID
182     }
183   });
184   if (!!result.errors || !result.data.disciplineOffer) {
185     yield put(errorPlan("Erro durante o carregamento da oferta de
      ↪  disciplina"));
186     return;
187   }
188   yield put(loadedDisciplineOffer(result.data.disciplineOffer));
189 }
190
191 export default function* planSaga<Shape>(client: ApolloClient<Shape>) {
192   yield takeEvery(addVersion, addVersionSaga, client);
193   yield takeEvery(loadDisciplineAction, loadDisciplineSaga, client);
194   yield takeEvery(loadDisciplineOfferAction, loadDisciplineOfferSaga,
    ↪  client);
195   yield takeEvery(LOCATION_CHANGE, loadPlanSaga, client);
```

```
196   yield call(loadPlanSaga, client);
197 }
```

Arquivo selectors.ts

```
1  import {
2    PlanVersion,
3    FullState,
4    Plan,
5    PlanPossibility,
6    Team,
7    DisciplineOfferReference,
8    SliceState,
9    BaseVersionPossibility,
10   LoadStatus,
11   DisciplineReference,
12   DisciplineSelection,
13   DisciplineOfferSelection,
14   TeamSelection,
15   DisciplineOfferTeam,
16   PlanUniversity,
17   DisciplinesMap
18 } from "./types";
19 import { createSelector } from "reselect";
20 import { Schedule as CalendarSchedule } from
   ↪ "../../../base/components/calendar";
21 import { getQueryString, QueryString } from
   ↪ "../../../base/store/querystring";
22 import {
23   RemotePlanInput,
24   RemotePlanDisciplineInput,
25   RemotePlanDisciplineOfferInput,
26   RemotePlanPossibilityInput
27 } from "../../../../graphql";
28 import { RouterRootState } from "connected-react-router";
29 import { CombinatorIndex } from
   ↪ "../combinations/combinator/combinations";
30 import { DisciplineOfferCardModel } from
   ↪ "../../../base/components/cards/discipline-offer";
31 import { ForeignKey } from "../../../../service-worker/database";
```

```
import { omit } from "../../../base/components/form";

function getSlice(state: FullState): SliceState {
  return state.plan;
}

function getPlanSlice(state: SliceState): Plan | null {
  return state.plan ? state.plan : null;
}

export function getPlanFull(state: FullState): Plan | null {
  return getPlanSlice(getSlice(state));
}

export function getLoadStatus(state: FullState): LoadStatus {
  let slice = getSlice(state);
  return slice.status;
}

export function getPlanID(state: FullState): string | null {
  let plan = getPlanFull(state);
  let result: string | null = null;
  if (plan) {
    result = plan.id;
  }
  return result;
}

export function getPlanLoadedVersionID(state: FullState): string | null {
  let plan = getPlanFull(state);
  let result: string | null = null;
  if (plan && plan.loadedVersion) {
    result = plan.loadedVersion.id;
  }
  return result;
}

export const getPlanTitle = createSelector(getPlanFull, function(
  plan: Plan | null
```

```
71  ): string {
72    let title = "Carregando...";
73    if (plan) {
74      title = plan.name ? plan.name + " - " : "";
75      title += plan.university.acronym + " - " + plan.period.name;
76    }
77    return title;
78  });
79
80  export const getPlanUniversity = createSelector(getPlanFull, function(
81    plan: Plan | null
82  ): PlanUniversity | null {
83    return plan ? plan.university : null;
84  });
85
86  export const getPlanPeriod = createSelector(getPlanFull, function(
87    plan: Plan | null
88  ): ForeignKey | null {
89    return plan ? plan.period : null;
90  });
91
92  export const getSelectedVersionIndex = createSelector(
93    getQueryString,
94    getPlanFull,
95    function(qs: QueryString, plan: Plan | null): string | null {
96      if (!plan) {
97        return null;
98      }
99      let param = qs["version"];
100     let { length } = plan.draftsOrder;
101     if (length) {
102       param = plan.draftsOrder[length - 1];
103     }
104     if (!param) {
105       param = plan.loadedVersion.id;
106       if (!param) {
107         throw new Error("Internal error: Plan doesn't have versions");
108       }
109     }
```

```
    return Array.isArray(param) ? param[0] : param;
  }
);

export const getSelectedVersion = createSelector(
  getSelectedVersionIndex,
  getPlanFull,
  function(
    selectedVersion: string | null,
    plan: Plan | null
  ): PlanVersion | null {
    if (
      !plan ||
      !selectedVersion ||
      (plan.loadedVersion.id !== selectedVersion &&
        !plan.drafts[selectedVersion])
    ) {
      return null;
    }
    return plan.loadedVersion.id === selectedVersion
      ? plan.loadedVersion
      : plan.drafts[selectedVersion];
  }
);

export const getLatestVersion = createSelector(getPlanFull, function(
  plan: Plan | null
): PlanVersion | null {
  if (!plan) {
    return null;
  }
  if (plan.draftsOrder.length) {
    return plan.drafts[plan.draftsOrder[plan.draftsOrder.length - 1]];
  }
  return plan.loadedVersion;
});

export const getSelectedPossibilityIndex = createSelector(
  getQueryString,
```

```
149   getSelectedVersion,
150   function(qs: QueryString, version: PlanVersion | null): number | null {
151     if (!version) {
152       return null;
153     }
154     let param = qs["possibility"];
155     let result = parseInt(Array.isArray(param) ? param[0] : param);
156     if (isNaN(result) || result < 0 || result >=
      ↪  version.possibilities.length) {
157       result = version.defaultPossibility;
158     }
159     return result;
160   }
161 );
162
163 export const getSelectedPossibility = createSelector(
164   getSelectedVersion,
165   getSelectedPossibilityIndex,
166   function(
167     version: PlanVersion | null,
168     selectedPossibility: number | null
169   ): PlanPossibility | null {
170     return version && selectedPossibility !== null
171       ? version.possibilities[selectedPossibility]
172       : null;
173   }
174 );
175
176 export const getTeams = (state: FullState) => getSlice(state).teams;
177
178 export const getDisciplineOffers = (state: FullState) =>
    ↪  getSlice(state).offers;
179
180 export const getDisciplines = (state: FullState) =>
    ↪  getSlice(state).disciplines;
181
182 function getDisciplineOfferTeam(key?: "enabled" | "selected", value?:
    ↪  boolean) {
183   return function(
```

```
    possibility: PlanPossibility | null,
    teams: { [id: string]: { [version: string]: Team } },
    disciplineOffers: {
      [id: string]: { [version: string]: DisciplineOfferReference };
    }
  ): DisciplineOfferTeam[] {
    let result: DisciplineOfferTeam[] = [];
    if (possibility) {
      for (let {
        teamID,
        offerID,
        ...selection
      } of possibility.selectionTeams) {
        if (key && selection[key] !== value) {
          continue;
        }
        let { version, color } = possibility.offers[offerID];
        let discipline = disciplineOffers[offerID][version].discipline;
        let teamVersion = discipline.teamVersions[teamID];
        let team = teams[teamID][teamVersion];
        result.push({
          discipline,
          team,
          color,
          selection
        });
      }
    }
    return result;
  };
}

function getTeam(rows: DisciplineOfferTeam[]): { [id: string]:
↪ TeamSelection } {
  var teams: { [id: string]: TeamSelection } = {};
  for (let { team, selection, color } of rows) {
    teams[team.id] = { team, selection };
  }
  return teams;
```

```
}

function getDisciplineOffer(
  rows: DisciplineOfferTeam[]
): { [id: string]: DisciplineOfferSelection } {
  let disciplines: { [id: string]: DisciplineOfferSelection } = {};
  for (let { discipline: offer, selection } of rows) {
    let result = disciplines[offer.id] ?? {
      offer,
      selection
    };
    result.selection = {
      enabled: result.selection.enabled || selection.enabled,
      selected: result.selection.selected || selection.selected
    };
    disciplines[offer.id] = result;
  }
  return disciplines;
}

export const getPossibilityDisciplineOfferTeam = createSelector(
  getSelectedPossibility,
  getTeams,
  getDisciplineOffers,
  getDisciplineOfferTeam()
);

export const getEnabledDisciplineOfferTeam = createSelector(
  getSelectedPossibility,
  getTeams,
  getDisciplineOffers,
  getDisciplineOfferTeam("enabled", true)
);

export const getDisabledDisciplineOfferTeam = createSelector(
  getSelectedPossibility,
  getTeams,
  getDisciplineOffers,
  getDisciplineOfferTeam("enabled", false)
```

```
);

export const getSelectedDisciplineOfferTeam = createSelector(
  getSelectedPossibility,
  getTeams,
  getDisciplineOffers,
  getDisciplineOfferTeam("selected", true)
);

export const getNotSelectedDisciplineOfferTeam = createSelector(
  getSelectedPossibility,
  getTeams,
  getDisciplineOffers,
  getDisciplineOfferTeam("selected", false)
);

export const getPossibilityDisciplineOffersVersions = createSelector(
  getSelectedPossibility,
  function(
    possibility: PlanPossibility | null
  ): { [disciplineOfferId: string]: string } {
    let offers: { [disciplineOfferId: string]: string } = {};
    if (!possibility) {
      return offers;
    }
    for (let [id, { version }] of Object.entries(possibility.offers)) {
      offers[id] = version;
    }
    return offers;
  }
);

export const getPossibilityDisciplineOffers = createSelector(
  getPossibilityDisciplineOfferTeam,
  getDisciplineOffer
);

export const getPossibilityDisciplineOfferCardModels = createSelector(
  getPossibilityDisciplineOffers,
```

```
300   getTeams,
301   function(
302     disciplines: {
303       [id: string]: DisciplineOfferSelection;
304     },
305     teams: {
306       [id: string]: { [version: string]: Team };
307     }
308   ): DisciplineOfferCardModel[] {
309     let results: DisciplineOfferCardModel[] = [];
310     for (let { offer } of Object.values(disciplines)) {
311       let { id, code, name, teamVersions } = offer;
312       let result: DisciplineOfferCardModel = {
313         id,
314         campus: offer.custom ? undefined : offer.campus,
315         code,
316         name,
317         teams: []
318       };
319       for (let [teamId, teamVersion] of Object.entries(teamVersions)) {
320         result.teams.push(teams[teamId][teamVersion]);
321       }
322       results.push(result);
323     }
324     return results;
325   }
326 );
327
328 export const getEnabledDisciplineOffers = createSelector(
329   getEnabledDisciplineOfferTeam,
330   getDisciplineOffer
331 );
332
333 export const getDisabledDisciplineOffers = createSelector(
334   getDisabledDisciplineOfferTeam,
335   getDisciplineOffer
336 );
337
338 export const getSelectedDisciplineOffers = createSelector(
```

```
339   getSelectedDisciplineOfferTeam,
340   getDisciplineOffer
341 );
342
343 export const getNotSelectedDisciplineOffers = createSelector(
344   getNotSelectedDisciplineOfferTeam,
345   getDisciplineOffer
346 );
347
348 export const getPossibilityTeamsVersions = createSelector(
349   getDisciplineOffers,
350   getSelectedPossibility,
351   function(
352     offers: { [id: string]: { [version: string]: DisciplineOfferReference
      ↪  } },
353     possibility: PlanPossibility | null
354   ): { [teamId: string]: string } {
355     let result: { [teamId: string]: string } = {};
356     if (!possibility) {
357       return result;
358     }
359     for (let [id, { version }] of Object.entries(possibility.offers)) {
360       let offer = offers[id][version].discipline;
361       result = Object.assign(result, offer.teamVersions);
362     }
363     return result;
364   }
365 );
366
367 export const getPossibilityTeams = createSelector(
368   getPossibilityDisciplineOfferTeam,
369   getTeam
370 );
371
372 export const getEnabledTeams = createSelector(
373   getEnabledDisciplineOfferTeam,
374   getTeam
375 );
376
```

```
377 export const getDisabledTeams = createSelector(
378   getDisabledDisciplineOfferTeam,
379   getTeam
380 );
381
382 export const getSelectedTeams = createSelector(
383   getSelectedDisciplineOfferTeam,
384   getTeam
385 );
386
387 export const getPossibilityDisciplines = createSelector(
388   getSelectedPossibility,
389   getDisciplines,
390   function(
391     possibility: PlanPossibility | null,
392     disciplines: DisciplinesMap
393   ): { [id: string]: DisciplineSelection } {
394     let result: { [id: string]: DisciplineSelection } = {};
395     if (possibility === null) {
396       return result;
397     }
398     for (let { disciplineID, enabled } of
      ↪  possibility.selectionDisciplines) {
399       let version = possibility.disciplines[disciplineID];
400       result[disciplineID] = {
401         discipline: disciplines[disciplineID][version].discipline,
402         selection: { enabled }
403       };
404     }
405     return result;
406   }
407 );
408
409 export function getPossibilityDisciplinesVersions(
410   state: FullState & RouterRootState
411 ): { [disciplineId: string]: string } {
412   let possibility = getSelectedPossibility(state);
413   return possibility ? possibility.disciplines : {};
414 }
```

```
415
416 export const getEnabledTeamsGroupedByDisciplines = createSelector(
417   getEnabledTeams,
418   function(teams: { [id: string]: TeamSelection }): Team[][] {
419     let result: Team[][] = [];
420     let indexes: { [id: string]: number } = {};
421     for (let { team } of Object.values(teams)) {
422       let { id } = team.discipline;
423       if (typeof indexes[id] === "undefined") {
424         indexes[id] = result.push([]) - 1;
425       }
426       result[indexes[id]].push(team);
427     }
428     return result;
429   }
430 );
431
432 export const getSelectedTeamsIndexesGroupedByDisciplines =
    ↪  createSelector(
433   getEnabledTeamsGroupedByDisciplines,
434   getSelectedTeams,
435   function(
436     enabledTeams: Team[][],
437     selectedTeams: { [id: string]: TeamSelection }
438   ): CombinatorIndex[] {
439     let result: CombinatorIndex[] = [];
440     for (let { team } of Object.values(selectedTeams)) {
441       let mainIndex = enabledTeams.findIndex(
442         teams => teams[0].discipline.id === team.discipline.id
443       );
444       let subIndex = enabledTeams[mainIndex].findIndex(
445         ({ id }) => id === team.id
446       );
447       result.push({
448         mainIndex,
449         subIndex
450       });
451     }
452     return result;
```

```
453   }
454 );
455
456 export const getPossibilities = createSelector(getSelectedVersion,
    ↪  function(
457   version: PlanVersion | null
458 ): PlanPossibility[] {
459   return version ? version.possibilities : [];
460 });
461
462 export const getPossibilityDisciplineOfferColors = createSelector(
463   getPossibilityDisciplineOfferTeam,
464   function(offers: DisciplineOfferTeam[]) {
465     let result: { [offerId: string]: string } = {};
466     for (let offer of offers) {
467       result[offer.discipline.id] = offer.color;
468     }
469     return result;
470   }
471 );
472
473 export const getSelectedCalendarSchedules = createSelector(
474   getSelectedDisciplineOfferTeam,
475   function(rows: DisciplineOfferTeam[]): CalendarSchedule[] {
476     let schedules: CalendarSchedule[] = [];
477     for (let { team, discipline, color } of rows) {
478       for (let schedule of team.schedules) {
479         schedules.push({
480           ...schedule,
481           id: team.id,
482           title: discipline.code + " - " + discipline.name,
483           color
484         });
485       }
486     }
487     return schedules;
488   }
489 );
490
```

```
export const getBaseVersionPossibility = createSelector(
  getSelectedPossibilityIndex,
  getSelectedVersionIndex,
  function(
    possibility: number | null,
    baseVersion: string | null
  ): BaseVersionPossibility | null {
    if (possibility === null || baseVersion === null) {
      return null;
    }
    return {
      possibility,
      baseVersion
    };
  }
);

export const getRemotePlanInput = function(
  state: FullState & RouterRootState
): RemotePlanInput | undefined {
  let slice = getSlice(state);
  let plan = getPlanSlice(slice);
  if (!plan) {
    return undefined;
  }
  let version = plan.drafts[plan.draftsOrder[plan.draftsOrder.length -
  ↪ 1]];
  if (!version) {
    return undefined;
  }
  let disciplines: RemotePlanDisciplineInput[] = [];
  let offers: RemotePlanDisciplineOfferInput[] = [];
  let possibilities: RemotePlanPossibilityInput[] = [];
  let offerVersions = new Set<string>();
  let teamVersions = new Set<string>();
  let disciplineVersions = new Set<string>();
  for (let possibility of version.possibilities) {
    var possibilityResult: RemotePlanPossibilityInput = {
      name: possibility.name,
```

```
529       selectionTeams: [],
530       selectionDisciplines: []
531     };
532     for (let team of possibility.selectionTeams) {
533       let { offerID } = team;
534       let offer = possibility.offers[offerID];
535       let offerVersion = offer.version;
536       possibilityResult.selectionTeams.push({
537         ...omit(team, ["__typename"]),
538         color: offer.color,
539         offerVersion
540       });
541
542       let offerKeyString = offerID + "|" + offerVersion;
543       if (!offerVersions.has(offerKeyString)) {
544         offerVersions.add(offerKeyString);
545         let offer = slice.offers[offerID][offerVersion];
546         let offerInstance = offer.discipline;
547         let offerResult: RemotePlanDisciplineOfferInput = {
548           id: offerID,
549           version: offerVersion,
550           code: offerInstance.code,
551           name: offerInstance.name,
552           campus: offerInstance.custom
553             ? undefined
554             : omit(offerInstance.campus, ["__typename"]),
555           genericID: offerInstance.custom ? undefined :
           ↪  offerInstance.genericID,
556           teams: [],
557           custom: offer.custom
558         };
559         for (let teamKey of Object.entries(offerInstance.teamVersions)) {
560           let teamKeyString = teamKey.join("|");
561           if (!teamVersions.has(teamKeyString)) {
562             teamVersions.add(teamKeyString);
563             let [teamID, teamVersion] = teamKey;
564             let team = slice.teams[teamID][teamVersion];
565             offerResult.teams.push(
566               omit(team, ["__typename", "discipline", "custom"])
```

```
            );
          }
        }
        offers.push(offerResult);
      }
    }
    for (let selectionDiscipline of possibility.selectionDisciplines) {
      let { disciplineID } = selectionDiscipline;
      let disciplineVersion = possibility.disciplines[disciplineID];
      possibilityResult.selectionDisciplines.push({
        ...omit(selectionDiscipline, ["__typename"]),
        disciplineVersion
      });

      let key = disciplineID + "|" + disciplineVersion;
      if (!disciplineVersions.has(key)) {
        disciplineVersions.add(key);
        let { discipline } =
        ↪  slice.disciplines[disciplineID][disciplineVersion];
        disciplines.push(omit(discipline, ["__typename"]));
      }
    }
    possibilities.push(possibilityResult);
  }
  return {
    id: plan.id,
    name: plan.name,
    periodID: plan.period.id,
    public: plan.public,
    universityID: plan.university.id,
    selectedPossibility: version.defaultPossibility,
    possibilities,
    offers,
    disciplines
  };
};
```

Arquivo slice.ts

```
import { createSlice } from "@reduxjs/toolkit";
```

```
import * as reducers from "./reducers";
import {
  SliceState,
  LoadStatus,
  PayloadCustomDiscipline,
  PayloadCustomTeam
} from "./types";
import {
  wrapActionVersionBasedOnReducer,
  wrapActionBasedOnReducer
} from "./wrapper";
import cuid from "cuid";

const initialState: SliceState = {
  offers: {},
  teams: {},
  disciplines: {},
  plan: null,
  status: LoadStatus.LOADING
};

let { actions, reducer } = createSlice({
  initialState,
  name: "@@plans",
  reducers
});

export const addPossibility = wrapActionVersionBasedOnReducer(
  reducers.addPossibility,
  actions.addPossibility
);

export const selectDefaultPossibility = wrapActionVersionBasedOnReducer(
  reducers.selectDefaultPossibility,
  actions.selectDefaultPossibility
);

export const setPossibilityName = wrapActionVersionBasedOnReducer(
  reducers.setPossibilityName,
```

```
  actions.setPossibilityName
);

export const setActualPossibilityName = wrapActionVersionBasedOnReducer(
  reducers.setActualPossibilityName,
  actions.setActualPossibilityName
);

export const removePossibility = wrapActionVersionBasedOnReducer(
  reducers.removePossibility,
  actions.removePossibility
);

const addCustomDisciplineInternal = wrapActionVersionBasedOnReducer(
  reducers.addCustomDiscipline,
  actions.addCustomDiscipline
);

export function addCustomDiscipline(
  payload: Pick<PayloadCustomDiscipline, "code" | "name" | "team">
) {
  return addCustomDisciplineInternal({
    ...payload,
    id: cuid()
  });
}

addCustomDiscipline.toString = () =>
↪  actions.addCustomDiscipline.toString();
addCustomDiscipline.type = addCustomDiscipline.toString();

export const updateCustomDiscipline = wrapActionVersionBasedOnReducer(
  reducers.updateCustomDiscipline,
  actions.updateCustomDiscipline
);

export const removeDisciplineOffer = wrapActionVersionBasedOnReducer(
  reducers.removeDisciplineOffer,
  actions.removeDisciplineOffer
```

```
);

const addCustomTeamInternal = wrapActionVersionBasedOnReducer(
  reducers.addCustomTeam,
  actions.addCustomTeam
);

export function addCustomTeam(
  payload: Pick<PayloadCustomTeam, "code" | "offerID" | "schedules">
) {
  return addCustomTeamInternal({
    ...payload,
    teamID: cuid()
  });
}

addCustomTeam.toString = () => actions.addCustomTeam.toString();
addCustomTeam.type = addCustomTeam.toString();

export const updateCustomTeam = wrapActionVersionBasedOnReducer(
  reducers.updateCustomTeam,
  actions.updateCustomTeam
);

export const removeTeam = wrapActionVersionBasedOnReducer(
  reducers.removeTeam,
  actions.removeTeam
);

export const enableTeam = wrapActionVersionBasedOnReducer(
  reducers.enableTeam,
  actions.enableTeam
);

export const disableDisciplineOffer = wrapActionVersionBasedOnReducer(
  reducers.disableDisciplineOffer,
  actions.disableDisciplineOffer
);

```

```
export const enableDisciplineOffer = wrapActionVersionBasedOnReducer(
  reducers.enableDisciplineOffer,
  actions.enableDisciplineOffer
);

export const disableTeam = wrapActionVersionBasedOnReducer(
  reducers.disableTeam,
  actions.disableTeam
);

export const enableDiscipline = wrapActionVersionBasedOnReducer(
  reducers.enableDiscipline,
  actions.enableDiscipline
);

export const disableDiscipline = wrapActionVersionBasedOnReducer(
  reducers.disableDiscipline,
  actions.disableDiscipline
);

export const enableOnlyTeam = wrapActionVersionBasedOnReducer(
  reducers.enableOnlyTeam,
  actions.enableOnlyTeam
);

export const { loadedPlan, resetPlan, loadingPlan, errorPlan } = actions;

export const loadedDiscipline = wrapActionVersionBasedOnReducer(
  reducers.loadedDiscipline,
  actions.loadedDiscipline
);

export const loadedDisciplineOffer = wrapActionVersionBasedOnReducer(
  reducers.loadedDisciplineOffer,
  actions.loadedDisciplineOffer
);

export const removeDiscipline = wrapActionVersionBasedOnReducer(
  reducers.removeDiscipline,
```

```
  actions.removeDiscipline
);

export const selectTeams = wrapActionVersionBasedOnReducer(
  reducers.selectTeams,
  actions.selectTeams
);

export const setSelectedTeamsNoCommit = wrapActionBasedOnReducer(
  reducers.setSelectedTeams,
  actions.setSelectedTeams
);

export const setSelectedTeams = wrapActionVersionBasedOnReducer(
  reducers.setSelectedTeams,
  actions.setSelectedTeams
);

export const setPlanSettings = wrapActionVersionBasedOnReducer(
  reducers.setPlanSettings,
  actions.setPlanSettings
);

export { reducer };
```

Arquivo wrapper.ts

```
import { PayloadAction } from "@reduxjs/toolkit";
import {
  MetaVersion,
  PlanPossibility,
  PlanVersion,
  SliceState,
  BaseVersionPossibility,
  MetaAction,
  AddVersion
} from "./types";
import produce, { original } from "immer";
import cuid from "cuid";
import { addVersion } from "./actions";
```

```
14
15 export function wrapNewPlanVersion<T>(
16   reducer: (
17     version: PlanVersion,
18     action: PayloadAction<T, string, MetaAction>,
19     state: SliceState
20   ) => void
21 ): (state: SliceState, action: MetaVersion<T>) => void {
22   let newReducer = produce(reducer);
23   return function(
24     state: SliceState,
25     action: PayloadAction<T, string, MetaAction>
26   ) {
27     if (!state.plan) {
28       return;
29     }
30     let { plan } = state;
31     let { baseVersion, targetVersion } = action.meta;
32     let newVersion: PlanVersion = {
33       id: targetVersion,
34       savedAt: new Date(),
35       defaultPossibility: 0,
36       possibilities: []
37     };
38     if (baseVersion) {
39       let oldVersion =
40         plan.loadedVersion.id === baseVersion
41           ? plan.loadedVersion
42           : state.plan.drafts[baseVersion];
43       newVersion.defaultPossibility = oldVersion.defaultPossibility;
44       newVersion.possibilities =
45         original(oldVersion.possibilities) || oldVersion.possibilities;
46       newVersion = newReducer(newVersion, action, state);
47     } else {
48       reducer(newVersion, action, state);
49     }
50     state.plan.drafts[targetVersion] = newVersion;
51     state.plan.draftsOrder.push(targetVersion);
52   };
```

```
}

export function wrapPlanPossibility<T>(
  reducer: (
    possibility: PlanPossibility,
    action: PayloadAction<T>,
    version: PlanVersion,
    state: SliceState
  ) => void
): (state: SliceState, action: MetaVersion<T>) => void {
  return wrapNewPlanVersion(function(
    version: PlanVersion,
    action: MetaVersion<T>,
    state: SliceState
  ) {
    let { possibility } = action.meta;
    reducer(version.possibilities[possibility], action, version, state);
  });
}

export function wrapAction<T>(
  action: (payload: T) => PayloadAction<T>
): (payload: T, state: BaseVersionPossibility) => MetaVersion<T> {
  return function(payload: T, state: BaseVersionPossibility):
  ↪  MetaVersion<T> {
    return {
      ...action(payload),
      meta: {
        targetVersion: cuid(),
        ...state
      }
    };
  };
}

export function wrapActionVersion<T>(
  action: (payload: T) => PayloadAction<T>
): (payload: T) => PayloadAction<AddVersion<PayloadAction<T>>> {
  function actionCreator(
```

```
    payload: T
  ): PayloadAction<AddVersion<PayloadAction<T>>> {
    return {
      payload: {
        originalPayload: action(payload),
        targetVersion: cuid()
      },
      type: addVersion.toString()
    };
  }
  actionCreator.type = (action as any).type;
  actionCreator.toString = () => action.toString();
  return actionCreator;
}

export function wrapActionVersionBasedOnReducer<T>(
  _: (state: SliceState, action: MetaVersion<T>) => void,
  action: (payload: T) => PayloadAction<T>
) {
  // NOTE: this shouldn't be necessary, to be honest
  // but apparently the Typescript compiler has a hard time
  // trying to work with generic types to get the correct parameter type
  // for the payload on @reduxjs/toolkit even using meta property on
  // PayloadAction.. So this function basically serves to restore
  // the type information correctly to the action creator
  return wrapActionVersion<T>(action);
}

export function wrapActionBasedOnReducer<T>(
  _: (state: SliceState, action: MetaVersion<T>) => void,
  action: (payload: T) => PayloadAction<T>
) {
  return action;
}
```

B.2.79 Pasta src/js/pages/plan-details/store/ui

Arquivo actions.ts

```
import { UpdateView } from "./types";
import { createAction } from "@reduxjs/toolkit";

export const updateView =
  ↪  createAction<UpdateView>("@@plan-ui/update-view");
export const backView = createAction("@@plan-ui/back-view");
```

Arquivo hooks.ts

```
import { View } from "./types";
import { useDispatch } from "react-redux";
import { useCallback } from "react";
import { updateView } from "./actions";

export function useUILink(view: View) {
  let dispatch = useDispatch();
  return useCallback(
    function() {
      dispatch(updateView({ view }));
    },
    [dispatch]
  );
}
```

Arquivo index.ts

```
import { ISagaModule } from "redux-dynamic-modules-saga";
import { FullState } from "./types";

export * from "./selectors";
export * from "./types";
export * from "./slice";
export * from "./actions";
export * from "./hooks";
import { reducer as planUI } from "./slice";
import planUiSaga from "./sagas";

export default function getReduxModule(): ISagaModule<FullState> {
  return {
```

```
    id: "plan-ui",
    reducerMap: {
      planUI
    },
    sagas: [planUiSaga]
  };
}
```

Arquivo reducers.ts

```
import { SliceState, View } from "./types";
import { PayloadAction } from "@reduxjs/toolkit";
import { RemoteSearchFilter } from "../../../../graphql";

export function updateOldView(state: SliceState, action:
↪  PayloadAction<View>) {
  if (state.oldView === action.payload) {
    return;
  }
  state.oldView = action.payload;
}

export function search(state: SliceState, { payload }:
↪  PayloadAction<string>) {
  state.searchString = payload;
}

export function togglePrerequisites(state: SliceState) {
  switch (state.searchFilter) {
    case RemoteSearchFilter.All:
      state.searchFilter = RemoteSearchFilter.Courses;
      break;
    case RemoteSearchFilter.Courses:
      state.searchFilter = RemoteSearchFilter.All;
      break;
    case RemoteSearchFilter.None:
      state.searchFilter = RemoteSearchFilter.Prerequisites;
      break;
    case RemoteSearchFilter.Prerequisites:
      state.searchFilter = RemoteSearchFilter.None;
```

```
      break;
  }
}

export function toggleCourses(state: SliceState) {
  switch (state.searchFilter) {
    case RemoteSearchFilter.All:
      state.searchFilter = RemoteSearchFilter.Prerequisites;
      break;
    case RemoteSearchFilter.Courses:
      state.searchFilter = RemoteSearchFilter.None;
      break;
    case RemoteSearchFilter.None:
      state.searchFilter = RemoteSearchFilter.Courses;
      break;
    case RemoteSearchFilter.Prerequisites:
      state.searchFilter = RemoteSearchFilter.All;
      break;
  }
}
```

Arquivo sagas.ts

```
import {
  pushQueryString,
  redirect,
  getQueryString
} from "../../../base/store/querystring";
import { PayloadAction } from "@reduxjs/toolkit";
import { View, defaultView, UpdateView } from "./types";
import { select, put, takeEvery } from "redux-saga/effects";
import { updateOldView } from "./slice";
import { getPlanView, getOldPlanView } from "./selectors";
import { updateView, backView } from "./actions";

export function* updateViewSaga({ payload }: PayloadAction<UpdateView>) {
  let oldView: View = yield select(getPlanView);
  if (oldView.split("-")[0] !== payload.view.split("-")[0]) {
    yield put(updateOldView(oldView));
  }
```

```
  let { id, version } = yield select(getQueryString);
  yield put(
    redirect({
      url: "/planos/editar",
      querystring: { ...payload, id, version }
    })
  );
}

export function* backViewSaga() {
  let oldView: View | undefined = yield select(getOldPlanView);
  yield put(
    updateView({
      view: oldView || defaultView
    })
  );
}

export default function* planUiSaga() {
  yield takeEvery(updateView, updateViewSaga);
  yield takeEvery(backView, backViewSaga);
}
```

Arquivo selectors.ts

```
import { createSelector } from "reselect";
import { getQueryString } from "../../../base/store/querystring";
import { RouterRootState } from "connected-react-router";
import { View, Titles, FullState } from "./types";
import { RemoteSearchFilter } from "../../../../graphql";

function isViewValid(view: string): view is View {
  return Titles.hasOwnProperty(view);
}

export function getPlanView(state: RouterRootState): View {
  let { view } = getQueryString(state);
  if (Array.isArray(view)) {
    view = view[0];
  }
```

```
16    if (!view) {
17      let home: View = "calendar-week";
18      view = home;
19    }
20    if (isViewValid(view)) {
21      return view;
22    }
23    return "404";
24  }
25
26  export const getPlanViewTitle = createSelector(
27    getPlanView,
28    function getPlanTitle(view: View): string {
29      return Titles[view];
30    }
31  );
32
33  export function getOldPlanView(state: FullState): View | undefined {
34    return state.planUI.oldView;
35  }
36
37  export function getSearchString(state: FullState): string {
38    return state.planUI.searchString;
39  }
40
41  export function getSearchFilter(state: FullState): RemoteSearchFilter {
42    return state.planUI.searchFilter;
43  }
```

Arquivo slice.ts

```
1  import { createSlice } from "@reduxjs/toolkit";
2  import * as reducers from "./reducers";
3  import { RemoteSearchFilter } from "../../../../graphql";
4
5  let { actions, reducer } = createSlice({
6    name: "@@plan-ui",
7    initialState: {
8      searchString: "",
9      searchFilter: RemoteSearchFilter.None
```

```
  },
  reducers
});

export const {
  updateOldView,
  search,
  togglePrerequisites,
  toggleCourses
} = actions;
export { reducer };
```

Arquivo types.ts

```
import { RemoteSearchFilter } from "../../../../graphql";

interface Pages {
  "calendar-three-days": string;
  "calendar-day": string;
  "calendar-week": string;
  "search-discipline-offers": string;
  "search-disciplines": string;
  "search-teams": string;
  "add-possibility": string;
  "add-team": string;
  "add-discipline-offer": string;
  "edit-team": string;
  "edit-discipline-offer": string;
  "edit-possibility": string;
  combinations: string;
  "404": string;
  discipline: string;
  "discipline-offer": string;
  team: string;
  "list-disciplines": string;
  "list-discipline-offers": string;
  "list-teams": string;
  versions: string;
  teacher: string;
  possibilities: string;
```

```
27    settings: string;
28    "search-teachers": string;
29  }
30
31  export const Titles: Pages = {
32    "calendar-three-days": "Calendário",
33    "calendar-day": "Calendário",
34    "calendar-week": "Calendário",
35    possibilities: "Possibilidades",
36    combinations: "Combinações",
37    "search-discipline-offers": "Buscar Ofertas de Disciplinas",
38    "search-disciplines": "Buscar Disciplinas",
39    "search-teams": "Buscar Turmas",
40    "add-possibility": "Adicionar Possibilidade",
41    "edit-possibility": "Editar Possibilidade",
42    "add-discipline-offer": "Adicionar Oferta de Disciplina",
43    "add-team": "Adicionar Turma",
44    "edit-discipline-offer": "Editar Oferta de Disciplina",
45    "edit-team": "Editar Turma",
46    "404": "Página não encontrada",
47    "list-disciplines": "Disciplinas",
48    discipline: "Informações da disciplina",
49    "discipline-offer": "Informações da oferta de disciplina",
50    team: "Informações da turma",
51    "list-teams": "Turmas",
52    "list-discipline-offers": "Ofertas de Disciplinas",
53    versions: "Versões",
54    settings: "Configurações",
55    teacher: "Informações do professor",
56    "search-teachers": "Buscar professores"
57  };
58
59  export type View = keyof Pages;
60
61  export type Views = View[];
62
63  export type UpdateView = {
64    view: View;
65    [id: string]: string | string[];
```

```
};

export const defaultView: View = "calendar-week";

export interface SliceState {
  oldView?: View;
  searchString: string;
  searchFilter: RemoteSearchFilter;
}

export interface FullState {
  planUI: SliceState;
}
```

B.2.80 Pasta src/js/pages/plan-details/components/views/custom

Arquivo manage-discipline-offer.tsx

```
import React, { Fragment, useCallback } from "react";
import { useTextField, useSubmit } from
↪  "../../../../base/components/form";
import { TextField } from "@rmwc/textfield";
import { Grid, GridCell } from "@rmwc/grid";
import { getQueryString } from "../../../../base/store/querystring";
import { useSelector, useDispatch } from "react-redux";
import {
  getDisciplineOffers,
  getPossibilityDisciplineOffers,
  DisciplineOffer,
  updateCustomDiscipline,
  addCustomDiscipline,
  PayloadCustomDiscipline
} from "../../../store/plan";
import { Button } from "@rmwc/button";
import cuid from "cuid";
import { Snackbar } from "@rmwc/snackbar";
import { Dispatch } from "redux";
import PlanContainer from "../../container";
import { ManageTeamForm, ManageTeamFormFields } from "./manage-team";

```

```
type ManageDisciplineOfferFormProps = {
  discipline?: DisciplineOffer;
};

async function update(
  dispatch: Dispatch<any>,
  id: string,
  payload: Pick<PayloadCustomDiscipline, "code" | "name">
) {
  dispatch(
    updateCustomDiscipline({
      id,
      ...payload
    })
  );
  return Promise.resolve(1);
}

function ManageDisciplineOfferForm(props: ManageDisciplineOfferFormProps)
↪ {
  let { discipline } = props;
  let [code, codeField] = useTextField(discipline?.code ?? "");
  let [name, nameField] = useTextField(discipline?.name ?? "");
  let dispatch = useDispatch();
  let add = useCallback(
    async function(team: ManageTeamFormFields) {
      dispatch(
        addCustomDiscipline({
          code,
          name,
          team
        })
      );
      return <Fragment>Disciplina cadastrada com sucesso</Fragment>;
    },
    [dispatch, code, name]
  );
  const { onSubmit, submitted, error, clearError } = useSubmit(() =>
    discipline
```

```
      ? update(dispatch, discipline.id, { code, name })
      : Promise.reject()
  );
  if (submitted) {
    return (
      <p>
        Oferta de disciplina
        {discipline ? " editada " : " criada "}
        com sucesso
      </p>
    );
  }
  let fields = (
    <Fragment>
      <GridCell span={12} className="form-row">
        <TextField
          name="code"
          label="Código da Oferta de Disciplina"
          type="text"
          {...codeField}
        />
      </GridCell>
      <GridCell span={12} className="form-row">
        <TextField
          name="name"
          label="Nome da Oferta de Disciplina"
          type="text"
          {...nameField}
        />
      </GridCell>
    </Fragment>
  );
  if (!discipline) {
    return <ManageTeamForm submit={add}>{fields}</ManageTeamForm>;
  }
  return (
    <form onSubmit={onSubmit}>
      {fields}
      <Grid>
```

```
99          <GridCell span={12} className="form-row">
100           <Button type="submit" raised>
101             Editar
102           </Button>
103         </GridCell>
104       </Grid>
105       {error ? <Snackbar open onClose={clearError} message={error} /> :
          ↪  null}
106     </form>
107   );
108 }
109
110 type ManageDisciplineOfferProps = {
111   edit: boolean;
112 };
113
114 export default function ManageDisciplineOffer({
115   edit
116 }: ManageDisciplineOfferProps) {
117   let queryString = useSelector(getQueryString);
118   let id = queryString["offer_id"];
119   if (edit && typeof id !== "string") {
120     return (
121       <PlanContainer>
122         <p>Requisição inválida: Parâmetros inválidos foram informados</p>
123       </PlanContainer>
124     );
125   }
126   let disciplineOffers = useSelector(getPossibilityDisciplineOffers);
127   let formProps: ManageDisciplineOfferFormProps = {};
128   if (edit && typeof id === "string") {
129     let disciplineOffer = disciplineOffers[id];
130     if (!disciplineOffer) {
131       return (
132         <PlanContainer>
133           <p>Oferta de disciplina não encontrada</p>
134         </PlanContainer>
135       );
136     }
```

```
137     if (!disciplineOffer.offer.custom) {
138       return (
139         <PlanContainer>
140           <p>Esta oferta de disciplina não pode ser editada</p>
141         </PlanContainer>
142       );
143     }
144     formProps.discipline = disciplineOffer.offer;
145   }
146   return (
147     <PlanContainer>
148       <ManageDisciplineOfferForm {...formProps} />
149     </PlanContainer>
150   );
151 }
```

Arquivo manage-team.tsx

```
1  import React, {
2    useState,
3    useCallback,
4    MouseEvent,
5    ReactNode,
6    Fragment
7  } from "react";
8  import {
9    useTextField,
10   useSubmit,
11   useSelect
12 } from "../../../../base/components/form";
13 import { TextField } from "@rmwc/textfield";
14 import { Grid, GridCell } from "@rmwc/grid";
15 import { getQueryString } from "../../../../base/store/querystring";
16 import { useSelector, useDispatch } from "react-redux";
17 import {
18   Team,
19   getPossibilityTeams,
20   PayloadCustomTeam,
21   updateCustomTeam,
```

```
22    addCustomTeam,
23    getPossibilityDisciplineOffers
24  } from "../../../store/plan";
25  import { Button } from "@rmwc/button";
26  import { Snackbar } from "@rmwc/snackbar";
27  import { Dispatch } from "redux";
28  import PlanContainer from "../../container";
29  import { Typography } from "@rmwc/typography";
30  import {
31    DataTable,
32    DataTableContent,
33    DataTableHead,
34    DataTableRow,
35    DataTableCell,
36    DataTableBody
37  } from "@rmwc/data-table";
38  import {
39    formatDayOfWeek,
40    formatHourMinute,
41    getScheduleKey,
42    allWeekDays,
43    isBefore,
44    isEqual
45  } from "../../../../base/components/calendar/utilities";
46  import { IconButton } from "@rmwc/icon-button";
47  import { Schedule } from "../../../../base/components/calendar";
48  import { Select } from "@rmwc/select";
49  import { HourMinute } from "../../../../base/components/calendar/types";
50
51  export type ManageTeamFormFields = { code: string; schedules: Schedule[]
    ↪ };
52  type ManageTeamFormProps = {
53    team?: Team;
54    children?: ReactNode;
55    submit: (fields: ManageTeamFormFields) => Promise<{}>;
56  };
57
58  async function updateOrAdd(
59    dispatch: Dispatch<any>,
```

```
  offerID: string,
  teamID: string | undefined,
  payload: Pick<PayloadCustomTeam, "code" | "schedules">
) {
  if (teamID) {
    return dispatch(
      updateCustomTeam({
        ...payload,
        offerID,
        teamID
      })
    );
  }
  return dispatch(
    addCustomTeam({
      ...payload,
      offerID
    })
  );
}

export function ManageTeamForm(props: ManageTeamFormProps) {
  let { team } = props;
  let [code, codeField] = useTextField(team?.code ?? "");
  let [schedules, setSchedules] = useState<Schedule[]>(team?.schedules ??
  ↪  []);
  let [tmpScheduleIndex, setTmpScheduleIndex] = useState<number>(-1);
  let hoursMinutes: string[] = [];
  for (let hour = 0; hour < 24; hour++) {
    for (let minute = 0; minute < 60; minute += 15) {
      hoursMinutes.push(formatHourMinute({ hour, minute }));
    }
  }
  let tmpSchedule: Schedule | undefined =
    tmpScheduleIndex >= 0 && tmpScheduleIndex < schedules.length
      ? schedules[tmpScheduleIndex]
      : undefined;
  let [dayOfWeek, dayOfWeekAttrs] = useSelect(
    tmpSchedule ? formatDayOfWeek(tmpSchedule.dayOfWeek) : undefined
```

```
98    );
99    let [start, startAttrs] = useSelect(
100     tmpSchedule ? formatHourMinute(tmpSchedule.start) : undefined
101   );
102   let [end, endAttrs] = useSelect(
103     tmpSchedule ? formatHourMinute(tmpSchedule.end) : undefined
104   );
105   let addEdit = useCallback(
106     function(evt: MouseEvent) {
107       evt.preventDefault();
108       if (!start || !end || allWeekDays.indexOf(dayOfWeek) < 0) {
109         return;
110       }
111       let [startHourValue, startMinuteValue] = start.split(":");
112       let [endHourValue, endMinuteValue] = end.split(":");
113       let dayOfWeekValue = allWeekDays.indexOf(dayOfWeek) + 2;
114       let startHourMinute: HourMinute = {
115         hour: parseInt(startHourValue),
116         minute: parseInt(startMinuteValue)
117       };
118       let endHourMinute: HourMinute = {
119         hour: parseInt(endHourValue),
120         minute: parseInt(endMinuteValue)
121       };
122       if (isBefore(endHourMinute, startHourMinute)) {
123         let tmp = startHourMinute;
124         startHourMinute = endHourMinute;
125         endHourMinute = tmp;
126       } else if (isEqual(startHourMinute, endHourMinute)) {
127         return;
128       }
129       let newSchedule: Schedule = {
130         dayOfWeek: dayOfWeekValue,
131         start: startHourMinute,
132         end: endHourMinute
133       };
134       let tmpSchedules = schedules.slice();
135       if (!tmpSchedule) {
136         tmpSchedules.push(newSchedule);
```

```
      } else {
        tmpSchedules[tmpScheduleIndex] = newSchedule;
        setTmpScheduleIndex(-1);
      }
      setSchedules(tmpSchedules);
    },
    [tmpSchedule, schedules, dayOfWeek, start, end]
  );
  let cancel = useCallback(function(evt: MouseEvent) {
    evt.preventDefault();
    setTmpScheduleIndex(-1);
  }, []);

  let remove = useCallback(
    function(evt: MouseEvent, index: number) {
      evt.preventDefault();
      if (index >= 0 && index < schedules.length) {
        let newSchedules = schedules.slice();
        newSchedules.splice(index, 1);
        setSchedules(newSchedules);
      }
    },
    [schedules]
  );

  const { onSubmit, submitted, error, clearError } = useSubmit(() =>
    props.submit({ code, schedules })
  );

  if (submitted) {
    return <p>{submitted}</p>;
  }
  return (
    <form onSubmit={onSubmit}>
      <Grid>
        {props.children}
        <GridCell span={12} className="form-row">
          <TextField
            name="code"
```

```
176             label="Código da Turma"
177             type="text"
178             {...codeField}
179           />
180         </GridCell>
181         <GridCell span={12} className="form-row">
182           <Typography use="headline5">Horários:</Typography>
183         </GridCell>
184         <GridCell span={12} className="form-row">
185           <DataTable>
186             <DataTableContent>
187               <DataTableHead>
188                 <DataTableRow>
189                   <DataTableCell>Dia da semana</DataTableCell>
190                   <DataTableCell>Horário de início</DataTableCell>
191                   <DataTableCell>Horário de fim</DataTableCell>
192                   <DataTableCell></DataTableCell>
193                   <DataTableCell></DataTableCell>
194                 </DataTableRow>
195               </DataTableHead>
196               <DataTableBody>
197                 {schedules.map((schedule, index) => (
198                   <DataTableRow key={getScheduleKey(schedule)}>
199                     <DataTableCell>
200                       {formatDayOfWeek(schedule.dayOfWeek)}
201                     </DataTableCell>
202                     <DataTableCell>
203                       {formatHourMinute(schedule.start)}
204                     </DataTableCell>
205                     <DataTableCell>
206                       {formatHourMinute(schedule.end)}
207                     </DataTableCell>
208                     <DataTableCell>
209                       <IconButton
210                         icon="edit"
211                         onClick={evt => {
212                           evt.preventDefault();
213                           setTmpScheduleIndex(index);
214                         }}
```

```
                    />
                  </DataTableCell>
                  <DataTableCell>
                    <IconButton
                      icon="delete"
                      onClick={evt => remove(evt, index)}
                    />
                  </DataTableCell>
                </DataTableRow>
              ))}
              <DataTableRow>
                <DataTableCell>
                  <Select
                    enhanced
                    options={allWeekDays}
                    {...dayOfWeekAttrs}
                  />
                </DataTableCell>
                <DataTableCell>
                  <Select enhanced options={hoursMinutes}
                  ↪ {...startAttrs} />
                </DataTableCell>
                <DataTableCell>
                  <Select enhanced options={hoursMinutes} {...endAttrs}
                  ↪ />
                </DataTableCell>
                <DataTableCell>
                  <IconButton icon="save" onClick={addEdit} />
                </DataTableCell>
                {!!tmpSchedule ? (
                  <DataTableCell>
                    <IconButton icon="cancel" onClick={cancel} />
                  </DataTableCell>
                ) : null}
              </DataTableRow>
            </DataTableBody>
          </DataTableContent>
        </DataTable>
      </GridCell>
```

```
252         <GridCell span={12} className="form-row">
253           <Button type="submit" raised>
254             {team ? "Editar" : "Adicionar"}
255           </Button>
256         </GridCell>
257       </Grid>
258       {error ? <Snackbar open onClose={clearError} message={error} /> :
          ↪  null}
259     </form>
260   );
261 }
262
263 type ManageTeamProps = {
264   edit: boolean;
265 };
266
267 export default function ManageTeam({ edit }: ManageTeamProps) {
268   let queryString = useSelector(getQueryString);
269   let offerID = queryString["offer_id"];
270   let teamID = queryString["team_id"];
271   let offers = useSelector(getPossibilityDisciplineOffers);
272   let teams = useSelector(getPossibilityTeams);
273   let dispatch = useDispatch();
274
275   if (typeof offerID !== "string" || (edit && typeof teamID !==
      ↪  "string")) {
276     return (
277       <PlanContainer>
278         <p>Requisição inválida: Parâmetros inválidos foram informados</p>
279       </PlanContainer>
280     );
281   }
282   if (!offers[offerID].offer.custom) {
283     return (
284       <PlanContainer>
285         <p>Esta oferta de disciplina não pode ser editada.</p>
286       </PlanContainer>
287     );
288   }
```

```
289   let offerIDString = offerID;
290   let formProps: ManageTeamFormProps = {
291     async submit(fields) {
292       await updateOrAdd(dispatch, offerIDString, formProps.team?.id,
          ↪ fields);
293       return (
294         <Fragment>
295           Turma
296           {formProps.team ? " editada " : " criada "}
297           com sucesso
298         </Fragment>
299       );
300     }
301   };
302   if (edit && typeof teamID === "string") {
303     let team = teams[teamID];
304     if (!team || team.team.discipline.id !== offerID) {
305       return (
306         <PlanContainer>
307           <p>Turma não encontrada</p>
308         </PlanContainer>
309       );
310     }
311     if (!teams[teamID].team.custom) {
312       return (
313         <PlanContainer>
314           <p>Esta turma não pode ser editada.</p>
315         </PlanContainer>
316       );
317     }
318     formProps.team = team.team;
319   }
320   return (
321     <PlanContainer>
322       <ManageTeamForm {...formProps} />
323     </PlanContainer>
324   );
325 }
```

B.2.81 Pasta src/js/pages/plan-details/components/views/details

Arquivo discipline-offer.tsx

```tsx
import React, { Fragment, useCallback, MouseEvent } from "react";
import { useSelector, useDispatch } from "react-redux";
import { getQueryString } from "../../../../base/store/querystring";
import {
  getTeams,
  getPlanUniversity,
  getPlanPeriod,
  getDisciplineOffers,
  loadDisciplineOffer,
  enableDisciplineOffer,
  getPossibilityDisciplineOffers,
  disableDisciplineOffer,
  removeDisciplineOffer,
  getPossibilityTeams,
  disableTeam,
  enableTeam
} from "../../../store/plan";
import {
  RemoteDisciplineOfferDetails,
  DisciplineOfferDetails,
  DisciplineOfferDetailsModel,
  TeamDisciplineOfferDetailsModel
} from "../../../../discipline-offer/components";
import { RemoteGetDisciplineOfferQuery } from "../../../../../graphql";
import PlanContainer from "../../container";
import { TopBarContent } from
↪  "../../../../base/components/partials/top-bar";
import PlanTopBar from "../../top-bar";
import { TopAppBarActionItem } from "@rmwc/top-app-bar";
import { updateView } from "../../../store/ui";

type Offer = Exclude<
  RemoteGetDisciplineOfferQuery["disciplineOffer"],
  undefined
>;

```

```
type Teams = Offer["teams"];

export default function DisciplineOffer() {
  let dispatch = useDispatch();
  let teams = useSelector(getTeams);
  let offers = useSelector(getDisciplineOffers);
  let university = useSelector(getPlanUniversity);
  let period = useSelector(getPlanPeriod);
  let queryString = useSelector(getQueryString);
  let possibilityOffers = useSelector(getPossibilityDisciplineOffers);
  let offerId = queryString["offer_id"];

  let possibilityTeams = useSelector(getPossibilityTeams);
  let onAction = useCallback(
    function(
      offer: DisciplineOfferDetailsModel,
      team: TeamDisciplineOfferDetailsModel
    ) {
      dispatch(
        !possibilityTeams[team.id]
          ? loadDisciplineOffer(offer.id)
          : possibilityTeams[team.id].selection.enabled
          ? disableTeam({
              offerID: offer.id,
              teamID: team.id
            })
          : enableTeam({ offerID: offer.id, teamID: team.id })
      );
    },
    [dispatch, possibilityTeams]
  );

  let onDetails = useCallback(function(
    offer: DisciplineOfferDetailsModel,
    team: TeamDisciplineOfferDetailsModel
  ) {
    dispatch(
      updateView({
        view: "team",
```

```
75          offer_id: offer.id,
76          team_id: team.id
77        })
78      );
79    },
80    []);
81    let isEnabled = useCallback(
82      function(
83        offer: DisciplineOfferDetailsModel,
84        team: TeamDisciplineOfferDetailsModel
85      ) {
86        return possibilityTeams[team.id]
87          ? possibilityTeams[team.id].selection.enabled
88          : false;
89      },
90      [possibilityTeams]
91    );
92
93    let isSelected = useCallback(
94      function(
95        offer: DisciplineOfferDetailsModel,
96        team: TeamDisciplineOfferDetailsModel
97      ) {
98        return possibilityTeams[team.id]
99          ? possibilityTeams[team.id].selection.selected
100         : false;
101     },
102     [possibilityTeams]
103   );
104
105   let add = useCallback(
106     function(evt: MouseEvent) {
107       evt.preventDefault();
108       if (typeof offerId === "string" && !possibilityOffers[offerId]) {
109         dispatch(loadDisciplineOffer(offerId));
110       }
111     },
112     [dispatch, possibilityOffers, offerId]
113   );
```

```
  let remove = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (typeof offerId === "string" && !!possibilityOffers[offerId]) {
        dispatch(removeDisciplineOffer(offerId));
      }
    },
    [dispatch, possibilityOffers, offerId]
  );
  let edit = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
        typeof offerId === "string" &&
        !!possibilityOffers[offerId] &&
        possibilityOffers[offerId].offer.custom
      ) {
        dispatch(
          updateView({
            view: "edit-discipline-offer",
            offer_id: offerId
          })
        );
      }
    },
    [dispatch, possibilityOffers, offerId]
  );
  let toggle = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (typeof offerId === "string" && !!possibilityOffers[offerId]) {
        let { enabled } = possibilityOffers[offerId].selection;
        dispatch(
          enabled
            ? disableDisciplineOffer(offerId)
            : enableDisciplineOffer(offerId)
        );
      }
    },
```

```
153       [dispatch, possibilityOffers, offerId]
154     );
155     if (typeof offerId !== "string" || !period || !university) {
156       return (
157         <PlanContainer>
158           <p>
159             Requisição inválida: parâmetros requeridos não foram
                ↪  especificados
160           </p>
161         </PlanContainer>
162       );
163     }
164     let possibilityOffer = possibilityOffers[offerId];
165     if (!!possibilityOffer) {
166       let offer = offers[offerId][possibilityOffer.offer.version];
167       if (offer) {
168         let discipline = offer.discipline;
169         let offerTeams: Teams = [];
170         for (let [teamId, teamVersion] of Object.entries(
171           discipline.teamVersions
172         )) {
173           let team = teams[teamId][teamVersion];
174           offerTeams.push({
175             ...team,
176             vacancies: !team.custom ? team.vacancies : undefined
177           });
178         }
179         let { enabled } = possibilityOffer.selection;
180         return (
181           <Fragment>
182             <PlanTopBar>
183               <TopAppBarActionItem
184                 checked={enabled}
185                 onClick={toggle}
186                 icon={enabled ? "check_circle_outline" : "check_circle"}
187               />
188               {discipline.custom ? (
189                 <TopAppBarActionItem
190                   icon="edit"
```

```
191                 title="Editar oferta de disciplina"
192                 onClick={edit}
193               />
194             ) : null}
195             <TopAppBarActionItem
196               icon="delete"
197               title="Remover oferta de disciplina"
198               onClick={remove}
199             />
200           </PlanTopBar>
201           <TopBarContent>
202             <DisciplineOfferDetails
203               disciplineOffer={{
204                 ...discipline,
205                 university,
206                 period,
207                 teams: offerTeams
208               }}
209               isEnabled={isEnabled}
210               isSelected={isSelected}
211               onAction={onAction}
212               onDetails={onDetails}
213             />
214           </TopBarContent>
215         </Fragment>
216       );
217     }
218   }
219   return (
220     <Fragment>
221       <PlanTopBar>
222         <TopAppBarActionItem
223           icon="add"
224           title="Adicionar oferta de disciplina"
225           onClick={add}
226         />
227       </PlanTopBar>
228       <TopBarContent>
229         <RemoteDisciplineOfferDetails
```

```
            offerId={offerId}
            isEnabled={isEnabled}
            isSelected={isSelected}
            onAction={onAction}
            onDetails={onDetails}
          />
        </TopBarContent>
      </Fragment>
    );
}
```

Arquivo discipline.tsx

```
import React, { useCallback, Fragment, MouseEvent } from "react";
import { useSelector, useDispatch } from "react-redux";
import { getQueryString } from "../../../../base/store/querystring";
import {
  getPlanUniversity,
  getDisciplines,
  getPossibilityDisciplinesVersions,
  getPossibilityDisciplines,
  disableDiscipline,
  enableDiscipline,
  loadDiscipline,
  removeDiscipline
} from "../../../store/plan";
import {
  DisciplineDetails,
  RemoteDisciplineDetails
} from "../../../../discipline/components";
import PlanContainer from "../../container";
import { RemoteDisciplineSummaryFragment } from "../../../../../../graphql";
import { useDisciplineLink } from "../../link";
import { updateView } from "../../../store/ui";
import { TopBarContent } from
↪  "../../../../base/components/partials/top-bar";
import { TopAppBarActionItem } from "@rmwc/top-app-bar";
import PlanTopBar from "../../top-bar";

```

```
export default function Discipline() {
  let disciplines = useSelector(getDisciplines);
  let university = useSelector(getPlanUniversity);
  let queryString = useSelector(getQueryString);
  let possibilityDisciplines = useSelector(getPossibilityDisciplines);
  let dispatch = useDispatch();
  let link = useDisciplineLink();
  let useDetails = useCallback(
    function({ id }: RemoteDisciplineSummaryFragment) {
      return link(id);
    },
    [dispatch]
  );
  let disciplineId = queryString["discipline_id"];
  let add = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
        typeof disciplineId === "string" &&
        !possibilityDisciplines[disciplineId]
      ) {
        dispatch(loadDiscipline(disciplineId));
      }
    },
    [dispatch, possibilityDisciplines, disciplineId]
  );
  let remove = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
        typeof disciplineId === "string" &&
        !!possibilityDisciplines[disciplineId]
      ) {
        dispatch(removeDiscipline(disciplineId));
      }
    },
    [dispatch, possibilityDisciplines, disciplineId]
  );
  let toggle = useCallback(
```

```
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
        typeof disciplineId === "string" &&
        !!possibilityDisciplines[disciplineId]
      ) {
        let { enabled } = possibilityDisciplines[disciplineId].selection;
        dispatch(
          enabled
            ? disableDiscipline(disciplineId)
            : enableDiscipline(disciplineId)
        );
      }
    },
    [dispatch, possibilityDisciplines, disciplineId]
  );
  if (typeof disciplineId !== "string" || !university) {
    return (
      <PlanContainer>
        <p>
          Requisição inválida: parâmetros requeridos não foram
          ↪ especificados
        </p>
      </PlanContainer>
    );
  }
  let possibilityDiscipline = possibilityDisciplines[disciplineId];
  if (!!possibilityDiscipline) {
    let { version } = possibilityDiscipline.discipline;
    let { enabled } = possibilityDiscipline.selection;
    let { discipline } = disciplines[disciplineId][version];
    if (discipline) {
      return (
        <Fragment>
          <PlanTopBar>
            <TopAppBarActionItem
              checked={enabled}
              onClick={toggle}
              icon={enabled ? "check_circle_outline" : "check_circle"}
```

```
            />
            <TopAppBarActionItem
              icon="delete"
              title="Remover disciplina"
              onClick={remove}
            />
          </PlanTopBar>
          <TopBarContent>
            <DisciplineDetails
              discipline={{
                ...discipline,
                university
              }}
              useDetails={useDetails}
            />
          </TopBarContent>
        </Fragment>
      );
    }
  }
  return (
    <Fragment>
      <PlanTopBar>
        <TopAppBarActionItem
          icon="add"
          title="Adicionar disciplina"
          onClick={add}
        />
      </PlanTopBar>
      <TopBarContent>
        <RemoteDisciplineDetails
          disciplineId={disciplineId}
          useDetails={useDetails}
        />
      </TopBarContent>
    </Fragment>
  );
}
```

Arquivo teacher.tsx

```
import React, { useCallback } from "react";
import { useSelector, useDispatch } from "react-redux";
import { getQueryString } from "../../../../base/store/querystring";
import {
  getPlanUniversity,
  getPlanPeriod,
  loadDisciplineOffer,
  getPossibilityTeams,
  disableTeam,
  enableTeam
} from "../../../store/plan";
import PlanContainer from "../../container";
import {
  RemoteTeacherDetails,
  TeacherTeamDetailsModel,
  TeacherDetailsModel
} from "../../../../teacher/components";
import { updateView } from "../../../store/ui";

export default function Teacher() {
  let dispatch = useDispatch();
  let university = useSelector(getPlanUniversity);
  let period = useSelector(getPlanPeriod);
  let queryString = useSelector(getQueryString);
  let teacherId = queryString["teacher_id"];

  let teams = useSelector(getPossibilityTeams);
  let onAction = useCallback(
    function(teacher: TeacherDetailsModel, team: TeacherTeamDetailsModel)
    ↪ {
      dispatch(
        !teams[team.id]
          ? loadDisciplineOffer(team.discipline.id)
          : teams[team.id].selection.enabled
          ? disableTeam({
              offerID: team.discipline.id,
              teamID: team.id
```

```
            })
          : enableTeam({ offerID: team.discipline.id, teamID: team.id })
      );
    },
    [dispatch, teams]
  );

  let onDetails = useCallback(function(
    teacher: TeacherDetailsModel,
    team: TeacherTeamDetailsModel
  ) {
    dispatch(
      updateView({
        view: "team",
        offer_id: team.discipline.id,
        team_id: team.id
      })
    );
  },
  []);
  let isEnabled = useCallback(
    function(teacher: TeacherDetailsModel, team: TeacherTeamDetailsModel)
    ↪ {
      return teams[team.id] ? teams[team.id].selection.enabled : false;
    },
    [teams]
  );

  let isSelected = useCallback(
    function(teacher: TeacherDetailsModel, team: TeacherTeamDetailsModel)
    ↪ {
      return teams[team.id] ? teams[team.id].selection.selected : false;
    },
    [teams]
  );
  if (typeof teacherId !== "string" || !period || !university) {
    return (
      <PlanContainer>
        <p>
```

```
74            Requisição inválida: parâmetros requeridos não foram
                ↪  especificados
75          </p>
76        </PlanContainer>
77      );
78    }
79    return (
80      <PlanContainer>
81        <RemoteTeacherDetails
82          periodID={period.id}
83          teacherID={teacherId}
84          isEnabled={isEnabled}
85          isSelected={isSelected}
86          onAction={onAction}
87          onDetails={onDetails}
88        />
89      </PlanContainer>
90    );
91  }
```

Arquivo team.tsx

```
1  import React, { Fragment, useCallback, MouseEvent } from "react";
2  import { useSelector, useDispatch } from "react-redux";
3  import { getQueryString } from "../../../../base/store/querystring";
4  import {
5    TeamDetails,
6    RemoteTeamDetails,
7    TeamDetailsModel
8  } from "../../../../team/components";
9  import {
10   getTeams,
11   getPlanUniversity,
12   getPlanPeriod,
13   loadDisciplineOffer,
14   getPossibilityTeams,
15   disableTeam,
16   enableTeam,
17   removeTeam
```

```
} from "../../../store/plan";
import PlanContainer from "../../container";
import { TopBarContent } from
  ↪ "../../../../base/components/partials/top-bar";
import PlanTopBar from "../../top-bar";
import { TopAppBarActionItem } from "@rmwc/top-app-bar";
import { updateView } from "../../../store/ui";

export default function Team() {
  let teams = useSelector(getTeams);
  let university = useSelector(getPlanUniversity);
  let period = useSelector(getPlanPeriod);
  let queryString = useSelector(getQueryString);
  let possibilityTeams = useSelector(getPossibilityTeams);
  let dispatch = useDispatch();
  let teamId = queryString["team_id"];
  let offerId = queryString["offer_id"];
  let add = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (typeof teamId === "string" && !possibilityTeams[teamId]) {
        dispatch(loadDisciplineOffer(teamId));
      }
    },
    [dispatch, possibilityTeams, teamId]
  );
  let remove = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (typeof teamId === "string" && !!possibilityTeams[teamId]) {
        dispatch(removeTeam(teamId));
      }
    },
    [dispatch, possibilityTeams, teamId]
  );
  let edit = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
```

```
        typeof teamId === "string" &&
        !!possibilityTeams[teamId] &&
        possibilityTeams[teamId].team.custom
      ) {
        dispatch(
          updateView({
            view: "edit-team",
            offer_id: offerId,
            team_id: teamId
          })
        );
      }
    },
    [dispatch, possibilityTeams, offerId, teamId]
  );
  let toggle = useCallback(
    function(evt: MouseEvent) {
      evt.preventDefault();
      if (
        typeof offerId === "string" &&
        typeof teamId === "string" &&
        !!possibilityTeams[teamId]
      ) {
        let { enabled } = possibilityTeams[teamId].selection;
        let team = { offerID: offerId, teamID: teamId };
        dispatch(enabled ? disableTeam(team) : enableTeam(team));
      }
    },
    [dispatch, possibilityTeams, offerId, teamId]
  );
  let discipline = useCallback(
    function(discipline: TeamDetailsModel["discipline"]) {
      return function() {
        dispatch(
          updateView({
            view: "discipline-offer",
            offer_id: discipline.id
          })
        );
```

```
95        };
96      },
97      [dispatch]
98    );
99    if (
100     typeof teamId !== "string" ||
101     typeof offerId !== "string" ||
102     !period ||
103     !university
104   ) {
105     return (
106       <PlanContainer>
107         <p>
108           Requisição inválida: parâmetros requeridos não foram
              ↪ especificados
109         </p>
110       </PlanContainer>
111     );
112   }
113   let possibilityTeam = possibilityTeams[teamId];
114   if (!!possibilityTeam) {
115     let { version } = possibilityTeam.team;
116     let team = teams[teamId][version];
117     let { enabled } = possibilityTeam.selection;
118     return (
119       <Fragment>
120         <PlanTopBar>
121           <TopAppBarActionItem
122             checked={enabled}
123             onClick={toggle}
124             icon={enabled ? "check_circle_outline" : "check_circle"}
125           />
126           {team.custom ? (
127             <TopAppBarActionItem
128               icon="edit"
129               title="Editar turma"
130               onClick={edit}
131             />
132           ) : null}
```

```
133             <TopAppBarActionItem
134               icon="delete"
135               title="Remover turma"
136               onClick={remove}
137             />
138           </PlanTopBar>
139           <TopBarContent>
140             <TeamDetails
141               team={{
142                 ...team,
143                 university,
144                 period,
145                 campus: team.custom ? undefined : team.discipline.campus,
146                 course: team.custom ? undefined : team.course,
147                 vacancies: team.custom ? undefined : team.vacancies
148               }}
149               useDiscipline={discipline}
150             />
151           </TopBarContent>
152         </Fragment>
153       );
154     }
155     return (
156       <Fragment>
157         <PlanTopBar>
158           <TopAppBarActionItem
159             icon="add"
160             title="Adicionar oferta de disciplina"
161             onClick={add}
162           />
163         </PlanTopBar>
164         <TopBarContent>
165           <RemoteTeamDetails
166             offerId={offerId}
167             teamId={teamId}
168             useDiscipline={discipline}
169           />
170         </TopBarContent>
171       </Fragment>
```

```
172   );
173 }
```

B.2.82 Pasta src/js/pages/plan-details/components/views/lists

Arquivo list-discipline-offers.tsx

```
1  import React, { Fragment } from "react";
2  import { Grid, GridCell } from "@rmwc/grid";
3  import { useSelector } from "react-redux";
4  import { getPossibilityDisciplineOfferCardModels } from
   ↪  "../../../store/plan";
5  import { CalendarProvider } from "../../../../base/components/calendar";
6  import PlanDisciplineOfferCard from "../../cards/discipline-offer";
7  import PlanContainer from "../../container";
8  import { TopBarContent } from
   ↪  "../../../../base/components/partials/top-bar";
9  import PlanTopBar from "../../top-bar";
10 import { TopAppBarActionItem } from "@rmwc/top-app-bar";
11 import { useUILink } from "../../../store/ui";
12 import ListTabs from "../../tabs/list-tabs";
13 import { Button } from "@rmwc/button";
14
15 export default function DisciplineOffers() {
16   let disciplines = useSelector(getPossibilityDisciplineOfferCardModels);
17   let link = useUILink("add-discipline-offer");
18   let search = useUILink("search-discipline-offers");
19   return (
20     <Fragment>
21       <PlanTopBar>
22         <TopAppBarActionItem
23           icon="add"
24           title="Adicionar oferta de disciplina"
25           onClick={link}
26         />
27       </PlanTopBar>
28       <TopBarContent>
29         <ListTabs />
30         <CalendarProvider width={344} height={194}>
31           <div className="results">
```

```
32              <Grid>
33                {disciplines.map(discipline => (
34                  <GridCell desktop={4} phone={4} tablet={4}
                    ↪  key={discipline.id}>
35                    <PlanDisciplineOfferCard item={discipline} />
36                  </GridCell>
37                ))}
38                {disciplines.length === 0 ? (
39                  <Fragment>
40                    <GridCell span={12}>
41                      Nenhuma oferta de disciplina encontrada. Use a busca
                        ↪  para
42                      adicionar uma nova oferta de disciplina, clicando no
                        ↪  botão
43                      abaixo.
44                    </GridCell>
45                    <GridCell span={12}>
46                      <Button onClick={search}>Buscar</Button>
47                    </GridCell>
48                  </Fragment>
49                ) : null}
50              </Grid>
51            </div>
52          </CalendarProvider>
53        </TopBarContent>
54      </Fragment>
55    );
56  }
```

Arquivo list-disciplines.tsx

```
1  import React, { Fragment } from "react";
2  import { Grid, GridCell } from "@rmwc/grid";
3  import { useSelector } from "react-redux";
4  import { getPossibilityDisciplines } from "../../../store/plan";
5  import PlanContainer from "../../container";
6  import PlanDisciplineCard from "../../cards/discipline";
7  import ListTabs from "../../tabs/list-tabs";
8  import { Button } from "@rmwc/button";
```

```
import { useUILink } from "../../../store/ui";

export default function Disciplines() {
  let disciplines =
  ↪  Object.values(useSelector(getPossibilityDisciplines));
  let search = useUILink("search-disciplines");
  return (
    <PlanContainer>
      <ListTabs />
      <div className="results">
        <Grid>
          {disciplines.map(({ discipline }) => (
            <GridCell desktop={4} phone={4} tablet={4}
            ↪  key={discipline.id}>
              <PlanDisciplineCard item={discipline} />
            </GridCell>
          ))}
          {disciplines.length === 0 ? (
            <Fragment>
              <GridCell span={12}>
                Nenhuma disciplina encontrada. Use a busca para adicionar
                ↪  uma
                nova disciplina, clicando no botão abaixo.
              </GridCell>
              <GridCell span={12}>
                <Button onClick={search}>Buscar</Button>
              </GridCell>
            </Fragment>
          ) : null}
        </Grid>
      </div>
    </PlanContainer>
  );
}
```

Arquivo list-teams.tsx

```
import React, { Fragment } from "react";
import { Grid, GridCell } from "@rmwc/grid";
```

```
import { useSelector } from "react-redux";
import { getPossibilityTeams } from "../../../store/plan";
import PlanContainer from "../../container";
import { CalendarProvider } from "../../../../base/components/calendar";
import PlanTeamCard from "../../cards/team";
import ListTabs from "../../tabs/list-tabs";
import { Button } from "@rmwc/button";
import { useUILink } from "../../../store/ui";

export default function Teams() {
  let teams = Object.values(useSelector(getPossibilityTeams));
  let search = useUILink("search-teams");
  return (
    <PlanContainer>
      <ListTabs />
      <CalendarProvider width={344} height={194}>
        <div className="results">
          <Grid>
            {teams.map(({ team }) => (
              <GridCell desktop={4} phone={4} tablet={4} key={team.id}>
                <PlanTeamCard item={team} />
              </GridCell>
            ))}
            {teams.length === 0 ? (
              <Fragment>
                <GridCell span={12}>
                  Nenhuma turma encontrada. Use a busca para adicionar
                  ↪ uma nova
                  turma, clicando no botão abaixo.
                </GridCell>
                <GridCell span={12}>
                  <Button onClick={search}>Buscar</Button>
                </GridCell>
              </Fragment>
            ) : null}
          </Grid>
        </div>
      </CalendarProvider>
    </PlanContainer>
```

```
41    );
42  }
```

B.2.83 Pasta src/js/pages/plan-details/components/views/possibilities

Arquivo list.tsx

```
1  import React, { MouseEvent, useCallback, Fragment } from "react";
2
3  import {
4    DataTable,
5    DataTableContent,
6    DataTableHead,
7    DataTableRow,
8    DataTableCell,
9    DataTableBody
10 } from "@rmwc/data-table";
11 import { useSelector, useDispatch } from "react-redux";
12 import { getPossibilities, removePossibility } from
   ↪  "../../../store/plan";
13 import { IconButton } from "@rmwc/icon-button";
14 import { updateView, useUILink } from "../../../store/ui";
15 import { TopBarContent } from
   ↪  "../../../../base/components/partials/top-bar";
16 import PlanTopBar from "../../top-bar";
17 import { TopAppBarActionItem } from "@rmwc/top-app-bar";
18
19 export default function ListPossibilities() {
20   let dispatch = useDispatch();
21   let possibilities = useSelector(getPossibilities);
22   let edit = useCallback(
23     function(evt: MouseEvent, possibility_id: string) {
24       evt.preventDefault();
25       dispatch(
26         updateView({
27           view: "edit-possibility",
28           possibility_id
29         })
30       );
31     },
```

```
32      [dispatch]
33    );
34    let remove = useCallback(
35      function(evt: MouseEvent, possibility_id: number) {
36        evt.preventDefault();
37        dispatch(removePossibility(possibility_id));
38      },
39      [dispatch]
40    );
41    let link = useUILink("add-possibility");
42    return (
43      <Fragment>
44        <PlanTopBar>
45          <TopAppBarActionItem
46            icon="add"
47            title="Adicionar oferta de disciplina"
48            onClick={link}
49          />
50        </PlanTopBar>
51        <TopBarContent>
52          <DataTable>
53            <DataTableContent>
54              <DataTableHead>
55                <DataTableRow>
56                  <DataTableCell>Nome</DataTableCell>{" "}
57                  <DataTableCell></DataTableCell>
58                  <DataTableCell></DataTableCell>
59                </DataTableRow>
60              </DataTableHead>
61              <DataTableBody>
62                {possibilities.map((possibility, index) => (
63                  <DataTableRow>
64                    <DataTableCell>{possibility.name}</DataTableCell>
65                    <DataTableCell>
66                      <IconButton
67                        icon="edit"
68                        onClick={evt => edit(evt, "" + index)}
69                      />
70                    </DataTableCell>
```

```
                    <DataTableCell>
                      <IconButton
                        icon="delete"
                        onClick={evt => remove(evt, index)}
                      />
                    </DataTableCell>
                  </DataTableRow>
                ))}
              </DataTableBody>
            </DataTableContent>
          </DataTable>
        </TopBarContent>
      </Fragment>
    );
}
```

Arquivo manage.tsx

```
import React, { Fragment, useState, MouseEvent } from "react";
import { Typography } from "@rmwc/typography";
import {
  useTextField,
  useSubmit,
  useCheckboxField
} from "../../../../base/components/form";
import { TextField } from "@rmwc/textfield";
import { ChipSet, Chip } from "@rmwc/chip";
import { GridCell, Grid } from "@rmwc/grid";
import {
  getPossibilities,
  PlanPossibility,
  setPossibilityName,
  addPossibility,
  getSelectedPossibilityIndex,
  getSelectedVersion
} from "../../../store/plan";
import { useSelector, useDispatch } from "react-redux";
import PlanContainer from "../../container";
import { Dispatch } from "redux";
```

```
import { Snackbar } from "@rmwc/snackbar";
import { getQueryString } from "../../../../base/store/querystring";
import { Switch } from "@rmwc/switch";
import { Button } from "@rmwc/button";

function choosePossibility(setCopyFrom: (id: number) => void, index:
↪  number) {
  return function(e: MouseEvent) {
    e.preventDefault();
    setCopyFrom(index);
  };
}

type Fields = {
  name: string;
  copyFrom: number;
  defaultPossibility: boolean;
};

async function submit(
  dispatch: Dispatch,
  props: ManagePossibilityFormProps,
  fields: Fields
) {
  if (props.possibilityId !== undefined) {
    return dispatch(
      setPossibilityName({
        id: props.possibilityId,
        name: fields.name,
        defaultPossibility: fields.defaultPossibility
      })
    );
  }
  return dispatch(addPossibility(fields));
}

type ManagePossibilityFormProps = {
  possibilityId?: number;
};
```

```
60
61 export function ManagePossibilityForm(props: ManagePossibilityFormProps)
   ↪ {
62   let version = useSelector(getSelectedVersion);
63   const possibilities: PlanPossibility[] = useSelector(getPossibilities);
64   let possibility =
65     props.possibilityId !== undefined
66       ? possibilities[props.possibilityId]
67       : undefined;
68   let [name, nameField] = useTextField(possibility?.name ?? "");
69   let [defaultPossibility, defaultPossibilityAttrs] = useCheckboxField(
70     version?.defaultPossibility === props.possibilityId
71   );
72   let [copyFrom, setCopyFrom] = useState<number>(-1);
73   let dispatch = useDispatch();
74   let selectedUndefined = copyFrom === -1;
75
76   const { onSubmit, submitted, error, clearError } = useSubmit(() =>
77     submit(dispatch, props, { name, copyFrom, defaultPossibility })
78   );
79   if (submitted) {
80     return <p>Possibilidade {possibility ? "editada" :
       ↪  "adicionada"}!</p>;
81   }
82   return (
83     <form onSubmit={onSubmit}>
84       <Grid>
85         <GridCell span={12} className="form-row">
86           <TextField name="name" label="Nome" type="text" {...nameField}
             ↪  />
87         </GridCell>
88         <GridCell span={12} className="form-row">
89           <Switch label="Possibilidade padrão?"
             ↪  {...defaultPossibilityAttrs} />
90         </GridCell>
91         {!possibility ? (
92           <Fragment>
93             <GridCell span={12} className="form-row">
94               <Typography use="headline5">Copiar de:</Typography>
```

```
95              </GridCell>
96              <GridCell span={12} className="form-row">
97                <ChipSet choice>
98                  <Chip
99                    aria-selected={selectedUndefined}
100                   selected={selectedUndefined}
101                   onClick={choosePossibility(setCopyFrom, -1)}
102                 >
103                   Nenhuma
104                 </Chip>
105                 {possibilities.map((possibility, index) => {
106                   const selected = index === copyFrom;
107                   return (
108                     <Chip
109                       role="option"
110                       aria-selected={selected}
111                       selected={selected}
112                       key={possibility.name}
113                       onClick={choosePossibility(setCopyFrom, index)}
114                     >
115                       {possibility.name}
116                     </Chip>
117                   );
118                 })}
119               </ChipSet>
120             </GridCell>
121           </Fragment>
122         ) : null}
123         <GridCell span={12} className="form-row">
124           <Button type="submit" raised>
125             {possibility ? "Editar" : "Adicionar"}
126           </Button>
127         </GridCell>
128       </Grid>
129       {error ? <Snackbar open onClose={clearError} message={error} /> :
        ↪  null}
130     </form>
131   );
132 }
```

```
133
134 type ManagePossibilityProps = {
135   edit: boolean;
136 };
137
138 export default function ManagePossibility({ edit }:
    ↪  ManagePossibilityProps) {
139   let queryString = useSelector(getQueryString);
140   let possibilities = useSelector(getPossibilities);
141   let id = queryString["possibility_id"];
142   if (edit && typeof id !== "string") {
143     return (
144       <PlanContainer>
145         <p>Requisição inválida: Parâmetros inválidos foram informados</p>
146       </PlanContainer>
147     );
148   }
149   let formProps: ManagePossibilityFormProps = {};
150   if (edit && typeof id === "string") {
151     let possibility = possibilities[parseInt(id)];
152     if (!possibility) {
153       return (
154         <PlanContainer>
155           <p>Possibilidade não encontrada</p>
156         </PlanContainer>
157       );
158     }
159     formProps.possibilityId = parseInt(id);
160   }
161   return (
162     <PlanContainer>
163       <ManagePossibilityForm {...formProps} />
164     </PlanContainer>
165   );
166 }
```

B.2.84 Pasta src/js/pages/plan-details/components/views/search

Arquivo search-discipline-offers.tsx

```
import React, { useCallback, useState, useEffect } from "react";
import { Grid, GridCell } from "@rmwc/grid";
import { CalendarProvider } from "../../../../base/components/calendar";
import { useQuery } from "@apollo/react-hooks";
import { useSelector } from "react-redux";
import { getSearchString, getSearchFilter } from "../../../store/ui";
import {
  searchDisciplineOffers,
  getAllDisciplineOffers
} from "../../../queries/search.gql";
import {
  RemoteSearchDisciplineOffersQuery,
  RemoteSearchDisciplineOffersQueryVariables
} from "../../../../../graphql";
import InfiniteScroll, {
  InfiniteScrollRenderOptions
} from "../../../../base/components/infinite-scroll";
import { getPlanPeriod } from "../../../store/plan";
import SearchString from "../../search";
import PlanContainer from "../../container";
import SearchTabs from "../../tabs/search-tabs";
import PlanDisciplineOfferCard from "../../cards/discipline-offer";
import SearchPrerequisites from "../../search-filter";

const defaultData: RemoteSearchDisciplineOffersQuery = {
  searchDisciplineOffers: {
    results: [],
    info: { query: { count: 0, version: "latest" }, page: { after: 0 } }
  }
};

type SearchDisciplineOffersRenderProps = InfiniteScrollRenderOptions<
  ↪ RemoteSearchDisciplineOffersQuery["searchDisciplineOffers"]["results"][0]
> & {
  searchString: string;
};

function SearchDisciplineOffersRender(
```

```
39    props: SearchDisciplineOffersRenderProps
40  ) {
41    let { items, ...searchProps } = props;
42    return (
43      <Grid>
44        <SearchString {...searchProps} />
45        {items.map(item => (
46          <GridCell key={item.id} desktop={4} tablet={4} phone={4}>
47            <PlanDisciplineOfferCard item={item} />
48          </GridCell>
49        ))}
50      </Grid>
51    );
52  }
53
54  export default function SearchDisciplineOffers() {
55    let searchString = useSelector(getSearchString);
56    let filter = useSelector(getSearchFilter);
57    let period = useSelector(getPlanPeriod);
58    let [version, setVersion] = useState<string>("latest");
59    let { loading, data, error, fetchMore } = useQuery<
60      RemoteSearchDisciplineOffersQuery,
61      RemoteSearchDisciplineOffersQueryVariables
62    >(searchString.length > 0 ? searchDisciplineOffers :
    ↪  getAllDisciplineOffers, {
63      variables: {
64        periodID: period?.id ?? "",
65        q: searchString,
66        version,
67        filter
68      }
69    });
70    let { results, info } = (data || defaultData).searchDisciplineOffers;
71    let { count } = info.query;
72    useEffect(
73      function() {
74        if (info.query.version === version) {
75          return;
76        }
```

```
      setVersion(info.query.version);
    },
    [info.query.version]
  );
  let fetchMoreCallback = useCallback(
    function() {
      fetchMore({
        variables: {
          offset: info.page.after
        },
        updateQuery(previous, { fetchMoreResult }) {
          let next = fetchMoreResult || defaultData;
          return {
            searchDisciplineOffers: {
              ...previous.searchDisciplineOffers,
              ...next.searchDisciplineOffers,
              results: [
                ...previous.searchDisciplineOffers.results,
                ...next.searchDisciplineOffers.results
              ]
            }
          };
        }
      });
    },
    [fetchMore, info.page.after]
  );
  if (error) {
    return (
      <PlanContainer>
        <div className="results">Erro ao carregar dados.</div>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <SearchTabs />
      <SearchPrerequisites />
      <div className="results">
```

```
        <CalendarProvider width={344} height={194}>
          <InfiniteScroll
            itemCount={count}
            loading={loading}
            loadMore={fetchMoreCallback}
            items={results}
            render={SearchDisciplineOffersRender}
            renderOptions={{ searchString }}
          />
        </CalendarProvider>
      </div>
    </PlanContainer>
  );
}
```

Arquivo search-disciplines.tsx

```
import React, { useCallback, useState, useEffect } from "react";
import { Grid, GridCell } from "@rmwc/grid";
import { CalendarProvider } from "../../../../base/components/calendar";
import { useQuery } from "@apollo/react-hooks";
import { useSelector, useDispatch } from "react-redux";
import { getSearchString, getSearchFilter } from "../../../store/ui";
import {
  searchDisciplines,
  getAllDisciplines
} from "../../../queries/search.gql";
import {
  RemoteSearchDisciplinesQuery,
  RemoteSearchDisciplinesQueryVariables
} from "../../../../../graphql";
import InfiniteScroll, {
  InfiniteScrollRenderOptions
} from "../../../../base/components/infinite-scroll";
import { getPlanUniversity } from "../../../store/plan";
import SearchString from "../../search";
import PlanContainer from "../../container";
import SearchTabs from "../../tabs/search-tabs";
import PlanDisciplineCard from "../../cards/discipline";
```

```
import SearchPrerequisites from "../../search-filter";

const defaultData: RemoteSearchDisciplinesQuery = {
  searchDisciplines: {
    results: [],
    info: { query: { count: 0, version: "latest" }, page: { after: 0 } }
  }
};

type SearchDisciplinesRenderProps = InfiniteScrollRenderOptions<
  RemoteSearchDisciplinesQuery["searchDisciplines"]["results"][0]
> & {
  searchString: string;
};

function SearchDisciplinesRender(props: SearchDisciplinesRenderProps) {
  let { items, ...searchProps } = props;
  return (
    <Grid>
      <SearchString {...searchProps} />
      {items.map(item => (
        <GridCell desktop={4} tablet={4} phone={4} key={item.id}>
          <PlanDisciplineCard item={item} />
        </GridCell>
      ))}
    </Grid>
  );
}

export default function SearchDisciplines() {
  let searchString = useSelector(getSearchString);
  let university = useSelector(getPlanUniversity);
  let [version, setVersion] = useState<string>("latest");
  let filter = useSelector(getSearchFilter);
  let { loading, data, fetchMore, error } = useQuery<
    RemoteSearchDisciplinesQuery,
    RemoteSearchDisciplinesQueryVariables
  >(searchString.length > 0 ? searchDisciplines : getAllDisciplines, {
    variables: {
```

```
      q: searchString,
      universityID: university?.id ?? "",
      version,
      filter
    }
  });
  let { results, info } = (data || defaultData).searchDisciplines;
  let { count } = info.query;
  useEffect(
    function() {
      if (info.query.version === version) {
        return;
      }
      setVersion(info.query.version);
    },
    [info.query.version]
  );
  let fetchMoreCallback = useCallback(
    function() {
      fetchMore({
        variables: {
          offset: info.page.after
        },
        updateQuery(previous, { fetchMoreResult }) {
          let next = fetchMoreResult || defaultData;
          return {
            searchDisciplines: {
              ...previous.searchDisciplines,
              ...next.searchDisciplines,
              results: [
                ...previous.searchDisciplines.results,
                ...next.searchDisciplines.results
              ]
            }
          };
        }
      });
    },
    [fetchMore, info.page.after]
```

```
  );
  if (error) {
    return (
      <PlanContainer>
        {" "}
        <div className="results">Erro ao carregar dados.</div>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <SearchTabs />
      <SearchPrerequisites />
      <div className="results">
        <CalendarProvider width={344} height={194}>
          <InfiniteScroll
            itemCount={count}
            loading={loading}
            loadMore={fetchMoreCallback}
            items={results}
            render={SearchDisciplinesRender}
            renderOptions={{ searchString }}
          />
        </CalendarProvider>
      </div>
    </PlanContainer>
  );
}
```

Arquivo search-teachers.tsx

```
import React, { useCallback, useState, useEffect } from "react";
import { Grid, GridCell } from "@rmwc/grid";
import { CalendarProvider } from "../../../../base/components/calendar";
import { useQuery } from "@apollo/react-hooks";
import { useSelector } from "react-redux";
import { getSearchString } from "../../../store/ui";
import { searchTeachers, getAllTeachers } from
↪  "../../../queries/search.gql";
```

```
import {
  RemoteSearchTeachersQuery,
  RemoteSearchTeachersQueryVariables
} from "../../../../../graphql";
import InfiniteScroll, {
  InfiniteScrollRenderOptions
} from "../../../../base/components/infinite-scroll";
import { getPlanPeriod } from "../../../store/plan";
import SearchString from "../../search";
import PlanContainer from "../../container";
import SearchTabs from "../../tabs/search-tabs";
import TeacherCard from "../../../../base/components/cards/teacher";
import { useTeacherLink } from "../../link";

const defaultData: RemoteSearchTeachersQuery = {
  searchTeachers: {
    results: [],
    info: { query: { count: 0, version: "latest" }, page: { after: 0 } }
  }
};

type SearchTeachersRenderProps = InfiniteScrollRenderOptions<
  RemoteSearchTeachersQuery["searchTeachers"]["results"][0]
> & {
  searchString: string;
};

function SearchTeachersRender(props: SearchTeachersRenderProps) {
  let { items, ...searchProps } = props;
  let link = useTeacherLink();
  return (
    <Grid>
      <SearchString {...searchProps} />
      {items.map(item => (
        <GridCell key={item.id} desktop={4} tablet={4} phone={4}>
          <TeacherCard
            teacher={{ ...item, teams: item.teams.results }}
            onDetails={link(item.id)}
          />
```

```
        </GridCell>
      ))}
    </Grid>
  );
}

export default function SearchTeachers() {
  let searchString = useSelector(getSearchString);
  let period = useSelector(getPlanPeriod);
  let [version, setVersion] = useState<string>("latest");
  let { loading, data, error, fetchMore } = useQuery<
    RemoteSearchTeachersQuery,
    RemoteSearchTeachersQueryVariables
  >(searchString.length > 0 ? searchTeachers : getAllTeachers, {
    variables: {
      periodID: period?.id ?? "",
      q: searchString,
      version
    }
  });
  let { results, info } = (data || defaultData).searchTeachers;
  let { count } = info.query;
  useEffect(
    function() {
      if (info.query.version === version) {
        return;
      }
      setVersion(info.query.version);
    },
    [info.query.version]
  );
  let fetchMoreCallback = useCallback(
    function() {
      fetchMore({
        variables: {
          offset: info.page.after
        },
        updateQuery(previous, { fetchMoreResult }) {
          let next = fetchMoreResult || defaultData;
```

```
          return {
            searchTeachers: {
              ...previous.searchTeachers,
              ...next.searchTeachers,
              results: [
                ...previous.searchTeachers.results,
                ...next.searchTeachers.results
              ]
            }
          };
        }
      });
    },
    [fetchMore, info.page.after]
  );
  if (error) {
    return (
      <PlanContainer>
        <div className="results">Erro ao carregar dados.</div>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <SearchTabs />
      <div className="results">
        <CalendarProvider width={344} height={194}>
          <InfiniteScroll
            itemCount={count}
            loading={loading}
            loadMore={fetchMoreCallback}
            items={results}
            render={SearchTeachersRender}
            renderOptions={{ searchString }}
          />
        </CalendarProvider>
      </div>
    </PlanContainer>
  );
```

```
}
```

Arquivo search-teams.tsx

```tsx
import React, { useCallback, useState, useEffect } from "react";
import { Grid, GridCell } from "@rmwc/grid";
import { CalendarProvider } from "../../../../base/components/calendar";
import { useQuery } from "@apollo/react-hooks";
import { useSelector } from "react-redux";
import { getSearchString, getSearchFilter } from "../../../store/ui";
import { searchTeams, getAllTeams } from "../../../queries/search.gql";
import {
  RemoteSearchTeamsQuery,
  RemoteSearchTeamsQueryVariables
} from "../../../../../../graphql";
import InfiniteScroll, {
  InfiniteScrollRenderOptions
} from "../../../../../base/components/infinite-scroll";
import { getPlanPeriod } from "../../../store/plan";
import SearchString from "../../search";
import PlanContainer from "../../container";
import SearchTabs from "../../tabs/search-tabs";
import PlanTeamCard from "../../cards/team";
import SearchPrerequisites from "../../search-filter";

const defaultData: RemoteSearchTeamsQuery = {
  searchTeams: {
    results: [],
    info: { query: { count: 0, version: "latest" }, page: { after: 0 } }
  }
};

type SearchTeamsRenderProps = InfiniteScrollRenderOptions<
  RemoteSearchTeamsQuery["searchTeams"]["results"][0]
> & {
  searchString: string;
};

function SearchTeamsRender(props: SearchTeamsRenderProps) {
  let { items, ...searchProps } = props;
```

```
37    return (
38      <Grid>
39        <SearchString {...searchProps} />
40        {items.map(item => (
41          <GridCell key={item.id} desktop={4} tablet={4} phone={4}>
42            <PlanTeamCard item={item} />
43          </GridCell>
44        ))}
45      </Grid>
46    );
47  }
48  export default function SearchTeams() {
49    let period = useSelector(getPlanPeriod);
50    let searchString = useSelector(getSearchString);
51    let filter = useSelector(getSearchFilter);
52    let [version, setVersion] = useState<string>("latest");
53    let { loading, data, error, fetchMore } = useQuery<
54      RemoteSearchTeamsQuery,
55      RemoteSearchTeamsQueryVariables
56    >(searchString.length > 0 ? searchTeams : getAllTeams, {
57      variables: {
58        periodID: period?.id ?? "",
59        q: searchString,
60        version,
61        filter
62      }
63    });
64    let { results, info } = (data || defaultData).searchTeams;
65    let { count } = info.query;
66    useEffect(
67      function() {
68        if (info.query.version === version) {
69          return;
70        }
71        setVersion(info.query.version);
72      },
73      [info.query.version]
74    );
75    let fetchMoreCallback = useCallback(
```

```
    function() {
      fetchMore({
        variables: {
          offset: info.page.after
        },
        updateQuery(previous, { fetchMoreResult }) {
          let next = fetchMoreResult || defaultData;
          return {
            searchTeams: {
              ...previous.searchTeams,
              ...next.searchTeams,
              results: [
                ...previous.searchTeams.results,
                ...next.searchTeams.results
              ]
            }
          };
        }
      });
    },
    [fetchMore, info.page.after]
  );
  if (error) {
    return (
      <PlanContainer>
        <div className="results">Erro ao carregar dados.</div>
      </PlanContainer>
    );
  }
  return (
    <PlanContainer>
      <SearchTabs />
      <SearchPrerequisites />
      <div className="results">
        <CalendarProvider width={344} height={194}>
          <InfiniteScroll
            itemCount={count}
            loading={loading}
            loadMore={fetchMoreCallback}
```

```
            items={results}
            render={SearchTeamsRender}
            renderOptions={{ searchString }}
          />
        </CalendarProvider>
      </div>
    </PlanContainer>
  );
}
```

B.2.85 Pasta src/js/pages/plan-details/store/combinations/combinator

Arquivo array.test.ts

```
import { Combinator, Rule, ArrayCombinator } from "./combinations";
import { start } from "repl";

const max = 12;
const min = 1;
const data: number[][] = [];
let total = 0;
for (let i = 0; i < max; i++) {
  const discipline: number[] = [];
  const totalEnd = total + Math.min(max - i + 1, Math.ceil(max / 4));
  for (let j = total; j < totalEnd; j++) {
    discipline.push(j);
    total++;
  }
  data.push(discipline);
}
const rule = new Rule();

let combinator = new Combinator<number, any>(data, min, max, rule);

combinator.count().then(function(count) {
  let arr = new ArrayCombinator(combinator, count);
  const r = start("> ");
  r.context.arr = arr;
  r.context.combinator = combinator;
  r.context.prev = function() {
```

```
    combinator.prev().then(function(result) {
      console.log(result);
    });
  };
  r.context.next = function() {
    combinator.next().then(function(result) {
      console.log(result);
    });
  };
  r.context.getIndex = function(index) {
    arr.getIndex(index).then(function(count) {
      console.log(index, " => ", count);
    });
  };
});
```

Arquivo client.ts

```
import bind from "bind-decorator";
import Server from "worker-loader!./server";
import {
  CombinationOptions,
  CombinationRequest,
  CombinationResponse
} from "./types";
import {
  CombinatorIndex,
  CombinatorInterface,
  CombinatorResult
} from "./combinations";

interface CallbackSuccess<T> {
  "@@combination/response/success/init": void;
  "@@combination/response/success/item": CombinatorResult<T>;
  "@@combination/response/success/count": number;
  "@@combination/response/success/delete": void;
  "@@combination/response/success/set": void;
}

type Callback<T> = {
```

```
  [K in keyof CallbackSuccess<T>]?: (result: CallbackSuccess<T>[K]) =>
  ↪ void;
} & {
  "@@combination/response/failure": any;
};

export default class Combinator {
  private worker: Server;
  private callbacks: { [id: string]: (data: MessageEvent) => void };
  private id: number;
  constructor() {
    this.worker = new Server();
    this.worker.onmessage = this.receive;
    this.callbacks = {};
    this.id = 0;
  }
  public getCombinator<T>(
    data: T[][],
    options: CombinationOptions
  ): Promise<CombinatorHandler<T>> {
    let messageID = this.id++;
    let id = `${messageID}-${data.length}`;
    return new Promise((resolve, reject) => {
      this.callbacks[id] = (ev: MessageEvent) => {
        const data = ev.data as CombinationResponse<T>;
        if (data.messageID !== messageID) {
          return reject(new Error(`Invalid message ID detected`));
        }
        switch (data.type) {
          case "@@combination/response/failure":
            return reject(data.reason);
          case "@@combination/response/success/init":
            return resolve(
              new CombinatorHandler<T>(
                this,
                id,
                (callback: (ev: MessageEvent) => void): void => {
                  this.callbacks[id] = callback;
                },
```

```
                () => {
                  delete this.callbacks[id];
                }
              )
            );
          default:
            return reject(new Error(`Invalid type detected
            ↪  '${data.type}'`));
        }
      };
      this.postMessage({
        type: "@@combination/request/init",
        id,
        data,
        options,
        messageID
      });
    });
  }

  public postMessage<T>(request: CombinationRequest<T>): void {
    this.worker.postMessage(request);
  }

  @bind
  private receive(ev: MessageEvent) {
    const { id } = ev.data as CombinationResponse<any>;
    let callback = this.callbacks[id];
    callback(ev);
  }
}

export class CombinatorHandler<T> implements CombinatorInterface<T> {
  private parent: Combinator;
  private id: string;
  private messageID: number;
  private onDispose: null | (() => void);

  private callbacks: {
```

```
99      [K: number]: Callback<T>;
100   };
101
102   constructor(
103     parent: Combinator,
104     id: string,
105     callback: (receiver: (data: MessageEvent) => void) => void,
106     onDispose: () => void
107   ) {
108     this.parent = parent;
109     this.id = id;
110     this.messageID = 0;
111     this.callbacks = {};
112     this.onDispose = onDispose;
113     callback(this.receive);
114   }
115
116   @bind
117   private receive(ev: MessageEvent) {
118     let response = ev.data as CombinationResponse<T>;
119     const callback = this.callbacks[response.messageID][response.type];
120     delete this.callbacks[response.messageID];
121     switch (response.type) {
122       case "@@combination/response/failure":
123         return callback(response.reason);
124       case "@@combination/response/success/count":
125       case "@@combination/response/success/item":
126         return callback(response.result);
127       case "@@combination/response/success/set":
128       case "@@combination/response/success/init":
129         callback();
130     }
131   }
132
133   private post<K extends keyof CallbackSuccess<T>>(
134     responseType: K,
135     requestType:
136       | "@@combination/request/next"
137       | "@@combination/request/prev"
```

```
      | "@@combination/request/count"
      | "@@combination/request/delete"
      | "@@combination/request/set",
    data: CombinatorIndex[] = []
  ): Promise<CallbackSuccess<T>[K]> {
    return new Promise<CallbackSuccess<T>[K]>((resolve, reject) => {
      if (this.onDispose === null) {
        return reject(new Error(`${this.id} is already disposed`));
      }
      let messageID = this.messageID++;
      this.callbacks[messageID] = {
        [responseType]: resolve,
        "@@combination/response/failure": reject
      };
      this.parent.postMessage({
        type: requestType,
        id: this.id,
        messageID,
        data
      });
    });
  }

  public async dispose(): Promise<void> {
    if (!this.onDispose) {
      return;
    }
    await this.post(
      "@@combination/response/success/delete",
      "@@combination/request/delete"
    );
    this.onDispose();
    this.onDispose = null;
  }

  public prev(): Promise<CombinatorResult<T>> {
    return this.post(
      "@@combination/response/success/item",
      "@@combination/request/prev"
```

```
177       );
178     }
179
180     public next(): Promise<CombinatorResult<T>> {
181       return this.post(
182         "@@combination/response/success/item",
183         "@@combination/request/next"
184       );
185     }
186
187     public count(): Promise<number> {
188       return this.post(
189         "@@combination/response/success/count",
190         "@@combination/request/count"
191       );
192     }
193
194     public set(indexes: CombinatorIndex[]): Promise<void> {
195       return this.post(
196         "@@combination/response/success/set",
197         "@@combination/request/set",
198         indexes
199       );
200     }
201   }
```

Arquivo combinations.benchmark.ts

```
1  import { Suite } from "benchmark";
2  import { Combinator, Rule } from "./combinations";
3
4  const suite = new Suite("Combinator");
5
6  const max = 12;
7  const min = 1;
8  const data: number[][] = [];
9  let total = 0;
10 for (let i = 0; i < max; i++) {
11   const discipline: number[] = [];
12   const totalEnd = total + Math.min(max - i + 1, Math.ceil(max / 4));
```

```
13    for (let j = total; j < totalEnd; j++) {
14      discipline.push(j);
15      total++;
16    }
17    data.push(discipline);
18  }
19  const rule = new Rule();
20
21  interface Resolver {
22    resolve(): void;
23  }
24
25  const limit = 1000;
26
27  let combinator = new Combinator<number, any>(data, min, max, rule);
28
29  function setup() {
30    combinator = new Combinator<number, any>(data, min, max, rule);
31  }
32
33  combinator.count().then(result => {
34    console.log(`Há ${result} resultados de combinações`);
35  });
36
37  suite
38    .add(
39      "#count()",
40      function(callback: Resolver) {
41        combinator.count().then(result => {
42          callback.resolve();
43        });
44      },
45      {
46        defer: true,
47        setup
48      }
49    )
50    .add(
51      `#count(${limit})`,
```

```
    function(callback: Resolver) {
      combinator.count(limit).then(result => {
        callback.resolve();
      });
    },
    {
      defer: true,
      setup
    }
  )
  .add(
    "#next()",
    function(callback: Resolver) {
      combinator.next().then(() => {
        callback.resolve();
      });
    },
    {
      defer: true,
      setup
    }
  )
  .add(
    "#prev()",
    function(callback: Resolver) {
      combinator.prev().then(() => {
        callback.resolve();
      });
    },
    {
      defer: true,
      setup
    }
  )
  // add listeners
  .on("cycle", function(event) {
    console.log(String(event.target));
  })
  .on("complete", function() {
```

```
    console.log(
      "Fastest is " + suite.filter("fastest").map(result => result.name)
    );
  })
  // run async
  .run({ async: true });
```

Arquivo combinations.ts

```
import Mutex from "./mutex";

export interface CombinatorIndex {
  mainIndex: number;
  subIndex: number;
}

interface CombinatorState<F> extends CombinatorIndex {
  mainMinimumIndex: number;
  subLength: number;
  state: State;
  filter: F;
}

function resetState<F>(filter: F): CombinatorState<F> {
  return {
    mainMinimumIndex: 0,
    mainIndex: 0,
    subIndex: 0,
    subLength: 0,
    filter,
    state: State.INITIAL
  };
}

const enum Direction {
  Next = 1,
  Prev = -1
}

export interface RuleInterface<T, F> {
```

```
  create(): F;
  check(item: Readonly<T>, filter: Readonly<F>, isFinal: boolean):
  ↪ boolean;
  update(item: Readonly<T>, filter: Readonly<F>, newFilter: F): F;
}

export interface CombinatorResult<T> {
  indexes: CombinatorIndex[];
  results: T[];
}

export interface CombinatorInterface<T> {
  next(): Promise<CombinatorResult<T>>;
  prev(): Promise<CombinatorResult<T>>;
  count(limit: number): Promise<number>;
}

const enum State {
  INITIAL,
  CHECK_LAST_MAIN_INDEX,
  CHECK_LAST_SUB_INDEX,
  INCREMENT_LAST_SUB_INDEX,
  CHECK_MAIN_INDEX,
  CHECK_SUB_INDEX,
  RECURSE,
  INCREMENT_SUB_INDEX
}

export interface Timer {
  now(): number;
}

const defaultTimer: Timer =
  typeof performance === "undefined" ? Date : performance;

type LoopCallback = () => Promise<boolean>;

type Looper = (fn: LoopCallback) => void;

```

```
export function getLoop(
  timeout: number = 200,
  delay: number = 10,
  timer: Timer = defaultTimer
): Looper {
  return async function loop(fn: LoopCallback) {
    const start = timer.now();
    let schedule = true;
    while (schedule && timer.now() - start < timeout) {
      schedule = await fn();
    }
    if (schedule) {
      setTimeout(loop, delay, fn);
    }
  };
}

const defaultLoop = getLoop();

export class Combinator<T, F> implements CombinatorInterface<T> {
  private _counter: number;
  private _stack: CombinatorState<F>[];
  private _inputs: T[][];
  private _inputsLength: number;
  private _rule: RuleInterface<T, F>;
  private _loop: Looper;
  private _minItems: number;
  private _maxItems: number;
  private _numItems: number;
  private _mutex: Mutex;

  constructor(
    inputs: T[][],
    min: number,
    max: number,
    rule: RuleInterface<T, F>,
    selected: CombinatorIndex[] | undefined = undefined,
    loop: Looper = defaultLoop
  ) {
```

```
    this._counter = 0;
    this._minItems = min;
    this._maxItems = max;
    this._numItems = -1;
    let newInputs: T[][] = [];
    let tmp = rule.create();
    for (let oldList of inputs) {
      let newList: T[] = [];
      for (let item of oldList) {
        if (rule.check(item, tmp, false)) {
          newList.push(item);
        }
      }
      if (newList.length > 0) {
        newInputs.push(newList);
      }
    }
    this._inputs = newInputs;
    this._inputsLength = newInputs.length;
    this._rule = rule;
    this._loop = loop;
    this._mutex = new Mutex();
    this._stack = new Array(max);
    for (let c = 0; c < max; c++) {
      this._stack[c] = resetState(rule.create());
    }
    if (selected !== undefined) {
      this._set(selected);
    }
  }

  private _set(indexes: CombinatorIndex[]) {
    let { _stack: stack, _inputs: inputs } = this;
    let stackLength = stack.length;
    let indexesLength = indexes.length;
    if (indexesLength < this._minItems || indexesLength > this._maxItems)
    ↪ {
      throw new Error(
```

```
146         `The number of indexes (${indexesLength}) should be between min
            ↪  (${this._minItems}) and max (${this._maxItems}) items`
147       );
148     }
149     let newStack = new Array(stackLength);
150     let oldFilter = this._rule.create();
151     let set = new Set();
152     for (let c = 0; c < indexesLength; ++c) {
153       let item = indexes[c];
154       if (
155         item.mainIndex < 0 ||
156         item.mainIndex >= inputs.length ||
157         set.has(item.mainIndex)
158       ) {
159         throw new Error(
160           `The main index should be between zero and ${inputs.length}.
              ↪  And shouldn't be repeated.`
161         );
162       }
163       set.add(item.mainIndex);
164       let subLength = inputs[item.mainIndex].length;
165       if (item.subIndex < 0 || item.subIndex >= subLength) {
166         throw new Error(
167           `The sub-index should be between zero and ${subLength}`
168         );
169       }
170       let value = this._inputs[item.mainIndex][item.subIndex];
171       if (!this._rule.check(value, oldFilter, c + 1 === indexesLength)) {
172         throw new Error(`The indexes informed didn't pass the rule
            ↪  check`);
173       }
174       oldFilter = this._rule.update(value, oldFilter,
          ↪  this._rule.create());
175       newStack[c] = {
176         ...resetState(oldFilter),
177         ...item,
178         subLength
179       };
180     }
```

```
    for (let c = indexesLength; c < stackLength; c++) {
      newStack[c] = resetState(this._rule.create());
    }
    this._counter = indexesLength - 1;
    this._stack = newStack;
  }

  public async set(indexes: CombinatorIndex[]): Promise<void> {
    const release = await this._mutex.acquire();
    try {
      this._set(indexes);
    } finally {
      release();
    }
  }

  public async next(): Promise<CombinatorResult<T>> {
    return this._run(Direction.Next);
  }

  public async prev(): Promise<CombinatorResult<T>> {
    return this._run(Direction.Prev);
  }

  public async count(limit: number = Infinity): Promise<number> {
    const release = await this._mutex.acquire();
    const { _stack, _numItems, _counter } = this;
    this._numItems = this._maxItems;
    // Create a new, reseted stack, that can manage
    // the count process easily without changing the oldStack
    const length = _stack.length;
    this._stack = new Array(length);
    for (let c = 0, l = length; c < l; c++) {
      this._stack[c] = resetState(this._rule.create());
    }
    this._reset(Direction.Next);
    const result = await this._length(limit);
    this._numItems = _numItems;
    this._stack = _stack;
```

```
220       this._counter = _counter;
221       release();
222       return result;
223     }
224
225     private _reset(direction: Direction) {
226       this._counter = 0;
227       this._stack[0] = resetState(this._rule.create());
228       this._stack[0].mainIndex =
229         direction === Direction.Next ? 0 : this._inputsLength -
          ↪  this._numItems;
230     }
231
232     private _advanceCombinator(direction: Direction) {
233       this._numItems -= direction;
234       if (this._numItems > this._maxItems) {
235         this._numItems = this._minItems;
236       } else if (this._numItems < this._minItems) {
237         this._numItems = this._maxItems;
238       }
239     }
240
241     private _getResult(): CombinatorResult<T> {
242       const results: T[] = new Array(this._counter + 1);
243       const indexes = new Array(this._counter + 1);
244       for (let c = 0; c <= this._counter; c++) {
245         let { mainIndex, subIndex } = this._stack[c];
246         indexes[c] = { mainIndex, subIndex };
247         results[c] = this._inputs[mainIndex][subIndex];
248       }
249       return {
250         indexes,
251         results
252       };
253     }
254
255     private _initialStep(direction: Direction, item: CombinatorState<F>) {
256       // This state determines if item is the last item of the inputs
```

```
257     // and, if it is, it sends it through state 1 so it can loop each
        ↪ subIndex
258     // and push the results
259     // if it isn't it starts to look into each array in the inputs
260     // sequentially
261     if (this._counter === this._numItems - 1) {
262       item.state = State.CHECK_LAST_MAIN_INDEX;
263     } else {
264       item.state = State.CHECK_MAIN_INDEX;
265     }
266   }
267   private _checkLastMainIndex(direction: Direction, item:
      ↪ CombinatorState<F>) {
268     // If item.mainIndex is still valid, find the right subIndex to
        ↪ analyze and
269     // jump into state 2
270     // if it isn't, stops the loop
271     if (
272       item.mainIndex >= item.mainMinimumIndex &&
273       item.mainIndex < this._inputsLength
274     ) {
275       item.subLength = this._inputs[item.mainIndex].length;
276       item.subIndex = direction === Direction.Prev ? item.subLength - 1 :
          ↪ 0;
277       item.state = State.CHECK_LAST_SUB_INDEX;
278     } else {
279       this._counter--;
280     }
281   }
282   private _checkLastSubIndex(
283     direction: Direction,
284     item: CombinatorState<F>
285   ): boolean {
286     // Check if the sub index is valid and, if it is, check if the input
        ↪ is valid
287     // and jump to the next jumpÍndex. If it isn't, just advance to the
        ↪ next mainIndex
288     // and jump to state 1 again
289     let result = false;
```

```
    if (item.subIndex >= 0 && item.subIndex < item.subLength) {
      item.state = State.INCREMENT_LAST_SUB_INDEX;
      result = this._rule.check(
        this._inputs[item.mainIndex][item.subIndex],
        item.filter,
        true
      );
    } else {
      // The loop finished, check the outer loop again
      item.mainIndex += direction;
      item.state = State.CHECK_LAST_MAIN_INDEX;
    }
    return result;
  }

  private _incrementLastSubIndex(
    direction: Direction,
    item: CombinatorState<F>
  ) {
    item.subIndex += direction;
    item.state = State.CHECK_LAST_SUB_INDEX;
  }

  private _checkMainIndex(direction: Direction, item: CombinatorState<F>)
  ↪ {
    if (
      item.mainIndex >= item.mainMinimumIndex &&
      item.mainIndex < this._inputsLength
    ) {
      item.subLength = this._inputs[item.mainIndex].length;
      item.subIndex = direction === Direction.Prev ? item.subLength - 1 :
      ↪  0;
      item.state = State.CHECK_SUB_INDEX;
    } else {
      this._counter--;
    }
  }

```

```
  private _checkSubIndex(direction: Direction, item: CombinatorState<F>)
  ↪ {
    if (item.subIndex >= 0 && item.subIndex < item.subLength) {
      if (
        this._rule.check(
          this._inputs[item.mainIndex][item.subIndex],
          item.filter,
          false
        )
      ) {
        item.state = State.RECURSE;
      } else {
        item.subIndex += direction;
      }
    } else {
      item.state = State.CHECK_MAIN_INDEX;
      item.mainIndex += direction;
    }
  }

  private _recurse(direction: Direction, item: CombinatorState<F>) {
    this._counter++;
    item.state = State.INCREMENT_SUB_INDEX;
    const newItem = this._stack[this._counter];
    newItem.filter = this._rule.update(
      this._inputs[item.mainIndex][item.subIndex],
      item.filter,
      newItem.filter
    );
    newItem.mainMinimumIndex = item.mainIndex + 1;
    newItem.mainIndex =
      direction === Direction.Prev
        ? this._inputsLength - 1
        : newItem.mainMinimumIndex;
    newItem.state = State.INITIAL;
  }

  private _incrementSubIndex(direction: Direction, item:
  ↪  CombinatorState<F>) {
```

```
    item.state = State.CHECK_SUB_INDEX;
    item.subIndex += direction;
  }

  private _step(direction: Direction, loops = 100): boolean {
    for (let i = 0; this._counter >= 0 && i < loops; i++) {
      const item = this._stack[this._counter];
      switch (item.state) {
        case State.INITIAL:
          this._initialStep(direction, item);
          loops++;
          break;
        case State.CHECK_LAST_MAIN_INDEX:
          this._checkLastMainIndex(direction, item);
          break;
        case State.CHECK_LAST_SUB_INDEX:
          return this._checkLastSubIndex(direction, item);
        case State.INCREMENT_LAST_SUB_INDEX:
          this._incrementLastSubIndex(direction, item);
          loops++;
          break;
        case State.CHECK_MAIN_INDEX:
          this._checkMainIndex(direction, item);
          break;
        case State.CHECK_SUB_INDEX:
          this._checkSubIndex(direction, item);
          break;
        case State.RECURSE:
          this._recurse(direction, item);
          break;
        case State.INCREMENT_SUB_INDEX:
          this._incrementSubIndex(direction, item);
          loops++;
          break;
        default:
          throw new Error("State not recognized: " + item.state);
      }
    }
    return false;
```

```
  }

  private _runMain(
    direction: Direction,
    actualItems: number
  ): Promise<CombinatorResult<T>> {
    let loopCount = 0;
    return new Promise((resolve, reject) => {
      this._loop(
        async (): Promise<boolean> => {
          if (this._counter < 0) {
            // What if we never find a valid result?
            // Need to reset counter
            this._advanceCombinator(direction);
            this._reset(direction);
            if (this._numItems === actualItems) {
              // Detected loop!
              if (loopCount > 1) {
                reject("No combinations found");
                return false;
              }
              loopCount++;
            }
          }
          let result: boolean;
          if ((result = this._step(direction))) {
            let r = this._getResult();
            resolve(r);
          }
          return !result;
        }
      );
    });
  }

  private _length(limit: number): Promise<number> {
    let count = 0;
    limit = Math.round(Math.abs(limit));
    return new Promise((resolve, reject) => {
```

```
      this._loop(
        async (): Promise<boolean> => {
          if (this._counter < 0) {
            // What if we never find a valid result?
            // Need to reset counter
            this._advanceCombinator(Direction.Next);
            this._reset(Direction.Next);
            if (this._numItems === this._maxItems) {
              // Lower the limit so the promise will be resolved anyway
              limit = -1;
            }
          }
          if (count >= limit) {
            resolve(count);
            return false;
          }
          if (this._step(Direction.Next)) {
            count++;
          }
          return true;
        }
      );
    });
  }

  private async _run(direction: Direction): Promise<CombinatorResult<T>>
  ↪ {
    const release = await this._mutex.acquire();
    if (this._numItems < 0) {
      this._numItems =
        direction === Direction.Next ? this._maxItems : this._minItems;
      this._reset(direction);
    }
    const result = await this._runMain(direction, this._numItems);
    release();
    return result;
  }
}

```

```
export class Rule implements RuleInterface<any, any> {
  public create(): any {
    return {};
  }
  public update(item: any, filter: any, newFilter: any): any {
    return newFilter;
  }
  public check(item: any, filter: any, isFinal: boolean): boolean {
    return true;
  }
}

interface ResultCallback<T> {
  resolve(arg: CombinatorResult<T>): void;
  promise: Promise<CombinatorResult<T>>;
}

export class ArrayCombinator<T> {
  private _combinator: CombinatorInterface<T>;
  private _index?: number;
  private _schedules: { [index: number]: ResultCallback<T> };
  private _processor: Mutex;
  private _scheduler: Mutex;
  private _loop: Looper;
  private _lastResult?: CombinatorResult<T>;
  public readonly length: number;

  constructor(
    combinator: CombinatorInterface<T>,
    length: number,
    loop: Looper = defaultLoop
  ) {
    this._combinator = combinator;
    this.length = length;
    this._loop = loop;
    this._processor = new Mutex();
    this._scheduler = new Mutex();
    this._schedules = {};
  }
```

```
518
519   public async getIndex(indexArg: number): Promise<CombinatorResult<T>> {
520     let { length, _schedules } = this;
521     const index = indexArg;
522     if (index < 0 || index >= length) {
523       throw new Error(`Index isn't valid. Should be between 0 and
      ↪  ${length}`);
524     }
525     if (index % 1 !== 0) {
526       throw new Error("Index isn't valid. It should be an integer.");
527     }
528     const release = await this._scheduler.acquire();
529     let promise: Promise<CombinatorResult<T>>;
530     if (!_schedules[index]) {
531       promise = new Promise<CombinatorResult<T>>(resolve => {
532         _schedules[index] = {
533           resolve,
534           promise
535         };
536       });
537       _schedules[index].promise = promise;
538     } else {
539       promise = _schedules[index].promise;
540     }
541     release();
542     const acquirer = this._processor.acquire();
543     const finishOrResult = await Promise.race([acquirer, promise]);
544     let result: CombinatorResult<T>;
545     if (typeof finishOrResult === "function") {
546       const { _index } = this;
547       if (_index === index && this._lastResult) {
548         /*
549          If we request the same index in a sequential way, like
550             await combinator.getIndex(1);
551             await combinator.getIndex(1);
552          for example, the index will be the same and we need
553          to use the last found value to be able to
554          answer this request */
555         result = this._lastResult;
```

```
        await this.resolveIndex(_index, result);
      } else {
        let direction: Direction;
        let half = length / 2;
        if (_index === undefined) {
          direction = half > index ? Direction.Next : Direction.Prev;
          this._index = direction === Direction.Prev ? 0 : length - 1;
        } else {
          let internal = Math.abs(_index - index) < half;
          let condition =
            (internal && _index < index) || (!internal && _index >
            ↪  index);
          direction = condition ? Direction.Next : Direction.Prev;
        }
        result = await this.run(direction, index);
      }
      this._lastResult = result;
      finishOrResult();
    } else {
      let finish = await acquirer;
      result = finishOrResult;
      finish();
    }
    return result;
  }

  public async *getSlice(start: number, end: number) {
    const increment = start < end ? 1 : -1;
    for (let c = start; c !== end; c += increment) {
      let result = await this.getIndex(c);
      yield result;
    }
  }

  private async resolveIndex(index: number, result: CombinatorResult<T>)
  ↪ {
    const releaseScheduler = await this._scheduler.acquire();
    this._schedules[index].resolve(result);
    delete this._schedules[index];
```

```
593       releaseScheduler();
594     }
595
596     private run(
597       direction: Direction,
598       index: number
599     ): Promise<CombinatorResult<T>> {
600       let { _loop, _index, _combinator, _schedules, length } = this;
601       const isNext = direction === Direction.Next;
602       return new Promise(resolve => {
603         _loop(
604           async (): Promise<boolean> => {
605             if (_index === undefined) {
606               throw new Error("Cannot call run without a base index");
607             }
608             let result = await (isNext ? _combinator.next() :
                ↪  _combinator.prev());
609             _index += direction;
610             if (_index < 0) {
611               _index = length - 1;
612             } else if (_index >= length) {
613               _index = 0;
614             }
615             if (!!_schedules[_index]) {
616               await this.resolveIndex(_index, result);
617             }
618             let condition = index !== _index;
619             this._index = _index;
620             if (!condition) {
621               resolve(result);
622             }
623             return condition;
624           }
625         );
626       });
627     }
628   }
```

Arquivo mutex.ts

```
type Pointer = {
  next?: Pointer;
  callback: () => void;
};
export default class Mutex {
  private _first?: Pointer;
  private _last?: Pointer;
  private _locked: boolean;
  constructor() {
    this._locked = false;
  }
  public acquire(): Promise<() => void> {
    return new Promise(resolve => {
      const release = () => {
        if (this._first) {
          const { callback, next } = this._first;
          this._first = next;
          callback();
        } else {
          this._locked = false;
          this._last = undefined;
        }
      };
      if (this._locked) {
        let item: Pointer = {
          callback() {
            resolve(release);
          }
        };
        if (!this._first) {
          this._first = item;
        } else if (this._last) {
          this._last.next = item;
        }
        this._last = item;
        return;
      }
      this._locked = true;
      resolve(release);
```

```
    });
  }
}
```

Arquivo rule.ts

```
import { RuleInterface } from "./combinations";
import { CombinationRuleOptions } from "./types";
import { Team, HourMinute } from "../../plan/types";
import { DayInterval } from "../../../../base/components/calendar/types";
import {
  toMinutes,
  toHourMinute,
  hasConflict,
  isEqual,
  compareSchedule
} from "../../../../base/components/calendar/utilities";

export interface Filter {
  schedules: DayInterval[];
  total: number;
}

function checkIgnore<
  T extends { [id: string]: boolean } & { [id: number]: boolean }
>(map: T | undefined, id: keyof T) {
  return map !== undefined && map[id] === false;
}

function calculateMaxMin(
  defaultValue: number,
  maxMin?: number,
  perClass?: number
) {
  return maxMin !== undefined && perClass !== undefined
    ? maxMin * perClass
    : defaultValue;
}

export default class Rule implements RuleInterface<Team, Filter> {
```

```
  private options: CombinationRuleOptions;
  private max: number;
  private min: number;

  constructor(options: CombinationRuleOptions) {
    this.options = options;
    this.max = calculateMaxMin(
      Infinity,
      options.maxClass,
      options.minutesPerClass
    );
    this.min = calculateMaxMin(
      -Infinity,
      options.minClass,
      options.minutesPerClass
    );
  }

  public create(): Filter {
    return {
      schedules: [],
      total: 0
    };
  }

  public check(
    item: Readonly<Team>,
    filter: Readonly<Filter>,
    isFinal: boolean
  ): boolean {
    let { options, max, min } = this;
    if (
      filter.total > max ||
      checkIgnore(options.teams, item.id) ||
      checkIgnore(options.disciplines, item.discipline.id)
    ) {
      return false;
    }
    for (let teacher of item.teachers) {
```

```
      if (checkIgnore(options.teachers, teacher.id)) {
        return false;
      }
    }
    let sum = 0;
    for (let schedule of item.schedules) {
      if (checkIgnore(options.days, schedule.dayOfWeek)) {
        return false;
      }
      const end = toMinutes(schedule.end);
      const start = toMinutes(schedule.start);
      for (let c = start; c < end; c += 60) {
        let hourMinute = toHourMinute(c);
        if (checkIgnore(options.hours, hourMinute.hour)) {
          return false;
        }
      }
      let conflict = filter.schedules.find(other =>
        hasConflict(other, schedule)
      );
      if (conflict) {
        return false;
      }
      sum += end - start;
    }
    let totalSum = filter.total + sum;
    return totalSum <= max && (!isFinal || totalSum >= min);
  }

  public update(
    item: Readonly<Team>,
    filter: Readonly<Filter>,
    newFilter: Filter
  ): Filter {
    newFilter.schedules = filter.schedules.concat(item.schedules);
    let sum = 0;
    for (let { start, end } of item.schedules) {
      sum += toMinutes(end) - toMinutes(start);
    }
```

```
    newFilter.total = filter.total + sum;
    return newFilter;
  }
}
```

Arquivo server.ts

```
import { Combinator, CombinatorResult } from "./combinations";
import { default as Rule, Filter } from "./rule";
import { CombinationRequest, CombinationResponse } from "./types";
import { Team } from "../../plan/types";

const combinations: {
  [K: string]: Combinator<any, Filter>;
} = {};

const worker = (self as any) as Worker;

function post<T>(response: CombinationResponse<T>) {
  worker.postMessage(response);
}

async function sendItemResponse<T>(
  id: string,
  messageID: number,
  response: Promise<CombinatorResult<T>>
): Promise<void> {
  const result = await response;
  post<T>({
    type: "@@combination/response/success/item",
    id,
    messageID,
    result
  });
}

interface Catchable {
  catch(callback: (reason: any) => void): void;
}

```

```
self.addEventListener("message", function(event: MessageEvent) {
  const request = event.data as CombinationRequest<Team>;
  const { id, messageID } = request;
  const target = { id, messageID };
  let result: Catchable | null = null;
  switch (request.type) {
    case "@@combination/request/count":
      result = combinations[id].count().then(function(result: number) {
        post({
          ...target,
          type: "@@combination/response/success/count",
          result
        });
      });
      break;
    case "@@combination/request/next":
      result = sendItemResponse(id, messageID, combinations[id].next());
      break;
    case "@@combination/request/prev":
      result = sendItemResponse(id, messageID, combinations[id].prev());
      break;
    case "@@combination/request/set":
      result = combinations[id].set(request.data).then(function() {
        return post({
          ...target,
          type: "@@combination/response/success/set"
        });
      });
      break;
    case "@@combination/request/delete":
      delete combinations[id];
      post({
        ...target,
        type: "@@combination/response/success/delete"
      });
      break;
    case "@@combination/request/init":
      if (combinations[id]) {
        post({
```

```
            ...target,
            type: "@@combination/response/failure",
            reason: `"${id}" already exists and cannot be initiated again.
            ↪  Call '#dispose()' on it first.`
          });
        } else {
          const { min, max, options } = request.options;
          combinations[id] = new Combinator(
            request.data,
            min,
            max,
            new Rule(options)
          );
          post({
            ...target,
            type: "@@combination/response/success/init"
          });
        }
        break;
    }
    if (result) {
      result.catch(function(reason: any) {
        post({
          ...target,
          type: "@@combination/response/failure",
          reason
        });
      });
    }
});
```

Arquivo types.ts

```
import { CombinatorIndex, CombinatorResult } from "./combinations";
import { Options } from "../types";

export interface CombinationBase {
  id: string;
  messageID: number;
}
```

```
8
9  export interface CombinationNext extends CombinationBase {
10   type: "@@combination/request/next";
11 }
12 export interface CombinationPrev extends CombinationBase {
13   type: "@@combination/request/prev";
14 }
15 export interface CombinationCount extends CombinationBase {
16   type: "@@combination/request/count";
17 }
18 export interface CombinationDelete extends CombinationBase {
19   type: "@@combination/request/delete";
20 }
21 export interface CombinationSet extends CombinationBase {
22   type: "@@combination/request/set";
23   data: CombinatorIndex[];
24 }
25
26 export type CombinationRuleOptions = Options;
27
28 export interface CombinationOptions {
29   min: number;
30   max: number;
31   options: CombinationRuleOptions;
32 }
33
34 export interface CombinationInit<T> extends CombinationBase {
35   type: "@@combination/request/init";
36   data: T[][];
37   options: CombinationOptions;
38 }
39
40 export type CombinationRequest<T> =
41   | CombinationNext
42   | CombinationPrev
43   | CombinationCount
44   | CombinationInit<T>
45   | CombinationDelete
46   | CombinationSet;
```

```
47
48 export interface CombinationFailure extends CombinationBase {
49   type: "@@combination/response/failure";
50   reason: any;
51 }
52
53 export interface CombinationSuccessCount extends CombinationBase {
54   type: "@@combination/response/success/count";
55   result: number;
56 }
57
58 export interface CombinationSuccessInit extends CombinationBase {
59   type: "@@combination/response/success/init";
60 }
61
62 export interface CombinationSuccessSet extends CombinationBase {
63   type: "@@combination/response/success/set";
64 }
65
66 export interface CombinationSuccessItem<T> extends CombinationBase {
67   type: "@@combination/response/success/item";
68   result: CombinatorResult<T>;
69 }
70
71 export interface CombinationSuccessDelete extends CombinationBase {
72   type: "@@combination/response/success/delete";
73 }
74
75 export type CombinationResponse<T> =
76   | CombinationFailure
77   | CombinationSuccessCount
78   | CombinationSuccessItem<T>
79   | CombinationSuccessInit
80   | CombinationSuccessDelete
81   | CombinationSuccessSet;
```

B.2.86 Pasta src/js/pages/plan-details/store/plan/reducers

Arquivo custom.ts

```
import { wrapPlanPossibility } from "../wrapper";
import {
  PlanPossibility,
  PayloadCustomDiscipline,
  PlanVersion,
  SliceState,
  PayloadCustomDisciplineUpdate,
  DisciplineOfferReference,
  PayloadCustomTeam,
  Team,
  PayloadTeamID
} from "../types";
import { PayloadAction } from "@reduxjs/toolkit";
import getColor from "../colors";
import produce, { original } from "immer";

function editDiscipline(
  version: PlanVersion,
  state: SliceState,
  possibility: PlanPossibility,
  disciplineID: string,
  update: (discipline: DisciplineOfferReference) => void
) {
  let oldDisciplineVersion = possibility.offers[disciplineID].version;
  let discipline = state.offers[disciplineID];
  let oldInstance = original(discipline[oldDisciplineVersion]);
  let custom = false;
  if (oldInstance && oldInstance.custom) {
    custom = true;
    let result = produce(oldInstance, update);
    if (result !== oldInstance) {
      discipline[version.id] = result;
    }
  }
  return custom;
}

function editTeam(
  version: PlanVersion,
```

```
  state: SliceState,
  possibility: PlanPossibility,
  ref: PayloadTeamID,
  update: (team: Team) => void
) {
  editDiscipline(version, state, possibility, ref.offerID, function(
    disciplineRef: DisciplineOfferReference
  ) {
    let oldTeamVersion =
    ↪  disciplineRef.discipline.teamVersions[ref.teamID];
    if (!state.teams[ref.teamID]) {
      state.teams[ref.teamID] = {};
    }
    let team = state.teams[ref.teamID];
    let old = original(team[oldTeamVersion]) || team[oldTeamVersion];
    let result = produce(old, update);
    team[version.id] = { ...result, version: version.id };
    disciplineRef.discipline.teamVersions[ref.teamID] = version.id;
    possibility.offers[ref.offerID].version = version.id;
  });
}

export const addCustomDiscipline = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload }: PayloadAction<PayloadCustomDiscipline>,
  version: PlanVersion,
  state: SliceState
) {
  let teamID = "team-" + payload.id;
  let offerID = "offer-" + payload.id;
  if (!state.offers[offerID]) {
    state.offers[offerID] = {};
  }
  if (!state.teams[teamID]) {
    state.teams[teamID] = {};
  }
  state.teams[teamID][version.id] = {
    ...payload.team,
    id: teamID,
```

```
    custom: true,
    discipline: {
      ...payload,
      id: offerID
    },
    tables: [],
    teachers: [],
    version: version.id
  };
  state.offers[offerID][version.id] = {
    custom: true,
    deleted: false,
    discipline: {
      ...payload,
      id: offerID,
      custom: true,
      version: version.id,
      teamVersions: {
        [teamID]: version.id
      }
    }
  };
  possibility.offers[offerID] = {
    version: version.id,
    color: getColor(possibility)
  };
  possibility.selectionTeams.push({
    enabled: true,
    offerID,
    teamID,
    selected: false
  });
});

export const updateCustomDiscipline = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload }: PayloadAction<PayloadCustomDisciplineUpdate>,
  version: PlanVersion,
  state: SliceState
```

```
) {
  editDiscipline(version, state, possibility, payload.id, function(
    disciplineRef: DisciplineOfferReference
  ) {
    if (payload.code) {
      disciplineRef.discipline.code = payload.code;
    }
    if (payload.name) {
      disciplineRef.discipline.name = payload.name;
    }
  });
});

export const addCustomTeam = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload }: PayloadAction<PayloadCustomTeam>,
  version: PlanVersion,
  state: SliceState
) {
  let { teamID, offerID, ...data } = payload;
  editDiscipline(version, state, possibility, offerID, function(
    disciplineRef: DisciplineOfferReference
  ) {
    disciplineRef.discipline.teamVersions[teamID] = version.id;
    if (!state.teams[teamID]) {
      state.teams[teamID] = {};
    }
    state.teams[teamID][version.id] = {
      ...data,
      id: teamID,
      custom: true,
      version: version.id,
      teachers: [],
      tables: [],
      discipline: {
        id: disciplineRef.discipline.id,
        code: disciplineRef.discipline.code,
        name: disciplineRef.discipline.name
      }
```

```
    };
    possibility.selectionTeams.push({
      offerID,
      teamID,
      enabled: true,
      selected: false
    });
  });
});

export const updateCustomTeam = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload }: PayloadAction<PayloadCustomTeam>,
  version: PlanVersion,
  state: SliceState
) {
  editTeam(version, state, possibility, payload, function(team: Team) {
    team.code = payload.code;
    team.schedules = payload.schedules;
  });
});
```

Arquivo index.ts

```
export * from "./custom";
export * from "./loader";
export * from "./possibility";
export * from "./remover";
export * from "./selection";
export * from "./status";
export * from "./plan";
```

Arquivo loader.ts

```
import { wrapPlanPossibility } from "../wrapper";

import {
  PlanPossibility,
  PayloadLoadedDisciplineOffer,
```

```
  PlanVersion,
  DisciplineOfferReference,
  PayloadLoadedDiscipline,
  SliceState
} from "..";

import { PayloadAction } from "@reduxjs/toolkit";

import getColor from "../colors";

import { RemoteLoadPlanVersionQuery } from "../../../../../graphql";

import { loadedPlan } from "./status";
import { loadPlan } from "../loader";

export const loadedDisciplineOffer = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload: offer }: PayloadAction<PayloadLoadedDisciplineOffer>,
  version: PlanVersion,
  state: SliceState
) {
  if (!state.offers[offer.id]) {
    state.offers[offer.id] = {};
  }
  if (!!state.offers[offer.id][offer.version]) {
    return;
  }
  let result: DisciplineOfferReference = {
    custom: false,
    deleted: false,
    discipline: {
      ...offer,
      custom: false,
      version: version.id,
      teamVersions: {}
    }
  };
  possibility.offers[offer.id] = {
    version: version.id,
```

```
    color: getColor(possibility)
  };
  for (let team of offer.teams) {
    if (!state.teams[team.id]) {
      state.teams[team.id] = {};
    }
    if (!state.teams[team.id][team.version]) {
      state.teams[team.id][team.version] = {
        ...team,
        discipline: result.discipline,
        custom: false,
        vacancies: team.vacancies ?? { filled: 0, offered: 0 }
      };
      result.discipline.teamVersions[team.id] = team.version;
    }
    possibility.selectionTeams.push({
      enabled: true,
      selected: false,
      offerID: offer.id,
      teamID: team.id
    });
  }
  state.offers[offer.id][version.id] = result;
});

export const loadedDiscipline = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload: discipline }: PayloadAction<PayloadLoadedDiscipline>,
  version: PlanVersion,
  state: SliceState
) {
  if (!state.disciplines[discipline.id]) {
    state.disciplines[discipline.id] = {};
  }
  if (!state.disciplines[discipline.id][discipline.version]) {
    state.disciplines[discipline.id][discipline.version] = {
      deleted: false,
      discipline
    };
```

```
  }
  possibility.disciplines[discipline.id] = discipline.version;
  possibility.selectionDisciplines.push({
    disciplineID: discipline.id,
    enabled: true
  });
});

export function resetPlan(
  state: SliceState,
  action: PayloadAction<RemoteLoadPlanVersionQuery>
) {
  state.disciplines = {};
  state.offers = {};
  state.teams = {};
  state.plan = null;
  loadedPlan(state);
  loadPlan(state, action.payload);
}
```

Arquivo plan.ts

```
import { wrapNewPlanVersion } from "../wrapper";
import { PlanVersion, PayloadPlanSettings, Plan, SliceState } from
↪  "../types";
import { PayloadAction } from "@reduxjs/toolkit";

export const setPlanSettings = wrapNewPlanVersion(function(
  _: PlanVersion,
  { payload }: PayloadAction<PayloadPlanSettings>,
  state: SliceState
) {
  if (!state.plan) {
    return;
  }
  state.plan.name = payload.name;
  state.plan.public = payload.public;
});
```

Arquivo possibility.ts

```
import { wrapNewPlanVersion, wrapPlanPossibility } from "../wrapper";

import { PlanVersion, PayloadCustomPossibility, PlanPossibility } from
↪  "..";

import { PayloadAction } from "@reduxjs/toolkit";
import { PayloadCreateCustomPossibility } from "../types";

export const addPossibility = wrapNewPlanVersion(function(
  version: PlanVersion,
  { payload }: PayloadAction<PayloadCreateCustomPossibility>
) {
  let newPossibility: PlanPossibility;
  let basicData = {
    name: payload.name
  };
  if (payload.copyFrom !== undefined && payload.copyFrom >= 0) {
    let possibility = version.possibilities[payload.copyFrom];
    if (possibility) {
      newPossibility = {
        ...possibility,
        ...basicData
      };
      version.possibilities.push(newPossibility);
    }
    return;
  }
  newPossibility = {
    ...basicData,
    selectionTeams: [],
    disciplines: {},
    offers: {},
    selectionDisciplines: []
  };
  version.possibilities.push(newPossibility);
  if (payload.defaultPossibility) {
    version.defaultPossibility = version.possibilities.length - 1;
  }
});
```

```
39
40 export const selectDefaultPossibility = wrapNewPlanVersion(
41   function selectDefaultPossibility(
42     state: PlanVersion,
43     { payload }: PayloadAction<number>
44   ) {
45     if (payload >= 0 && payload < state.possibilities.length) {
46       state.defaultPossibility = payload;
47     }
48   }
49 );
50
51 export const setPossibilityName = wrapNewPlanVersion(
52   function setPossibilityName(
53     version: PlanVersion,
54     { payload }: PayloadAction<PayloadCustomPossibility>
55   ) {
56     if (payload.id >= 0 && payload.id < version.possibilities.length) {
57       version.possibilities[payload.id].name = payload.name;
58       if (payload.defaultPossibility) {
59         version.defaultPossibility = payload.id;
60       }
61     }
62   }
63 );
64
65 export const setActualPossibilityName = wrapPlanPossibility(
66   function setActualPossibilityName(
67     possibility: PlanPossibility,
68     action: PayloadAction<string>
69   ) {
70     possibility.name = action.payload;
71   }
72 );
73
74 export const removePossibility = wrapNewPlanVersion(function(
75   version: PlanVersion,
76   { payload }: PayloadAction<number>
77 ) {
```

```
78    if (
79      payload >= 0 &&
80      payload < version.possibilities.length &&
81      version.possibilities.length > 1
82    ) {
83      version.possibilities.splice(payload, 1);
84    }
85  });
```

Arquivo remover.ts

```
1   import { wrapPlanPossibility } from "../wrapper";
2
3   import { PlanPossibility } from "..";
4
5   import { PayloadAction } from "@reduxjs/toolkit";
6
7   export const removeDisciplineOffer = wrapPlanPossibility(function(
8     possibility: PlanPossibility,
9     { payload: offerID }: PayloadAction<string>
10  ) {
11    possibility.selectionTeams = possibility.selectionTeams.filter(
12      selected => selected.offerID !== offerID
13    );
14  });
15
16  export const removeTeam = wrapPlanPossibility(function(
17    possibility: PlanPossibility,
18    { payload: teamID }: PayloadAction<string>
19  ) {
20    possibility.selectionTeams = possibility.selectionTeams.filter(selected
      ↪  => {
21      return selected.teamID !== teamID;
22    });
23  });
24
25  export const removeDiscipline = wrapPlanPossibility(function(
26    possibility: PlanPossibility,
27    { payload: disciplineID }: PayloadAction<string>
28  ) {
```

```
  possibility.selectionDisciplines =
  ↪ possibility.selectionDisciplines.filter(
    selected => selected.disciplineID !== disciplineID
  );
});
```

Arquivo selection.ts

```
import { PlanPossibility, PayloadTeamID } from "..";
import { PayloadAction } from "@reduxjs/toolkit";
import { wrapPlanPossibility } from "../wrapper";
import { Team, PayloadTeamSelection } from "../types";

function editTeamEnabled(
  possibility: PlanPossibility,
  action: PayloadAction<PayloadTeamID>,
  enabled: boolean
) {
  let { offerID: disciplineID, teamID } = action.payload;
  let teams = possibility.selectionTeams;
  for (let i = 0, l = teams.length; i < l; i++) {
    let selection = teams[i];
    if (selection.offerID === disciplineID && selection.teamID ===
    ↪ teamID) {
      teams[i] = {
        ...selection,
        enabled,
        selected: selection.selected && enabled
      };
    }
  }
  possibility.selectionTeams = teams;
}

export const enableTeam = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  action: PayloadAction<PayloadTeamID>
) {
  editTeamEnabled(possibility, action, true);
});
```

```
32
33 export const disableTeam = wrapPlanPossibility(function(
34   possibility: PlanPossibility,
35   action: PayloadAction<PayloadTeamID>
36 ) {
37   editTeamEnabled(possibility, action, false);
38 });
39
40 function editDisciplineOffer(
41   possibility: PlanPossibility,
42   offerID: string,
43   isEnabled: (teamID: string) => boolean
44 ) {
45   let teams = possibility.selectionTeams;
46   for (let i = 0, l = teams.length; i < l; i++) {
47     let selection = teams[i];
48     if (selection.offerID === offerID) {
49       let enabled = isEnabled(selection.teamID);
50       teams[i] = {
51         ...selection,
52         enabled,
53         selected: selection.selected && enabled
54       };
55     }
56   }
57   possibility.selectionTeams = teams;
58 }
59
60 export const enableDisciplineOffer = wrapPlanPossibility(function(
61   possibility: PlanPossibility,
62   { payload }: PayloadAction<string>
63 ) {
64   editDisciplineOffer(possibility, payload, () => true);
65 });
66
67 export const disableDisciplineOffer = wrapPlanPossibility(function(
68   possibility: PlanPossibility,
69   { payload }: PayloadAction<string>
70 ) {
```

```
  editDisciplineOffer(possibility, payload, () => false);
});

export const enableOnlyTeam = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  { payload }: PayloadAction<PayloadTeamID>
) {
  editDisciplineOffer(
    possibility,
    payload.offerID,
    teamID => teamID === payload.teamID
  );
});

function editDisciplineEnabled(
  possibility: PlanPossibility,
  { payload }: PayloadAction<string>,
  enabled: boolean
) {
  let disciplines = possibility.selectionDisciplines;
  for (let i = 0, l = disciplines.length; i < l; i++) {
    let selection = disciplines[i];
    if (selection.disciplineID === payload) {
      disciplines[i] = { ...selection, enabled };
    }
  }
  possibility.selectionDisciplines = disciplines;
}

export const enableDiscipline = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  action: PayloadAction<string>
) {
  editDisciplineEnabled(possibility, action, true);
});

export const disableDiscipline = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  action: PayloadAction<string>
```

```
) {
  editDisciplineEnabled(possibility, action, false);
});

export const selectTeams = wrapPlanPossibility(function(
  possibility: PlanPossibility,
  action: PayloadAction<PayloadTeamSelection>
) {
  let { selectionTeams } = possibility;
  let selections: { [offerID: string]: string } = {};
  for (let selection of action.payload) {
    selections[selection.offerID] = selection.teamID;
  }
  for (let i = 0, l = selectionTeams.length; i < l; i++) {
    let selection = selectionTeams[i];
    selectionTeams[i] = {
      ...selection,
      selected:
        !!selections[selection.offerID] &&
        selections[selection.offerID] === selection.teamID
    };
  }
});

export const setSelectedTeams = wrapPlanPossibility(function
↪ setSelectedTeams(
  possibility: PlanPossibility,
  action: PayloadAction<Team[]>
) {
  let selected: { [discipline: string]: string | undefined } = {};
  for (let team of action.payload) {
    selected[team.discipline.id] = team.id;
  }
  for (let i = 0, l = possibility.selectionTeams.length; i < l; i++) {
    let team = possibility.selectionTeams[i];
    possibility.selectionTeams[i] = {
      ...team,
      selected: selected[team.offerID] === team.teamID
    };
```

```
148   }
149 });
```

Arquivo status.ts

```
1  import { PayloadAction } from "@reduxjs/toolkit";
2
3  import { LoadStatus, SliceState } from "..";
4
5  export function loadingPlan(state: SliceState, _: PayloadAction<void>) {
6    state.status = LoadStatus.LOADING;
7  }
8
9  export function loadedPlan(state: SliceState) {
10   state.status = LoadStatus.LOADED;
11 }
12
13 export function errorPlan(
14   state: SliceState,
15   { payload }: PayloadAction<string>
16 ) {
17   state.status = LoadStatus.ERROR;
18   state.error = payload;
19 }
```

B.2.87 Pasta src/js/pages/plan-details/store/plan/selectors

Arquivo index.ts

```
1
```

B.2.88 Pasta src/js/pages/plan-details/store/plan/types

Arquivo discipline.ts

```
1  import { ForeignKey } from "./foreign_key";
2  import { Table } from "./table";
3  import {
4    RemoteDisciplineRelatedType,
5    RemoteDisciplineRelatedItemType
```

```
} from "../../../../../graphql";

export type DisciplineRelatedItemType = RemoteDisciplineRelatedItemType;

export type DisciplineRelatedItem = {
  type: DisciplineRelatedItemType;
  value: string;
  description?: string;
};

export type DisciplineRelatedType = RemoteDisciplineRelatedType;

export type DisciplineRelated = {
  name: string;
  type: DisciplineRelatedType;
  items: DisciplineRelatedItem[];
};

export type DisciplineRelatedSummary = {
  id: string;
  course: ForeignKey;
  habilitation: ForeignKey;
  step: ForeignKey;
  genericID: string;
  version: string;
  code: string;
  name: string;
  type: string;
  description: string;
};

export type Discipline = {
  readonly id: string;
  readonly version: string;
  course: ForeignKey;
  habilitation: ForeignKey;
  step: ForeignKey;
  genericID: string;
  code: string;
```

```
  name: string;
  type: string;
  description: string;
  related: DisciplineRelated[];
  relatedDisciplines: DisciplineRelatedSummary[];
  tables: Table[];
};

export type DisciplineReference = {
  discipline: Discipline;
  deleted: boolean;
  updatedVersion?: string;
};
```

Arquivo discipline_offer.ts

```
import { ForeignKey } from "./foreign_key";
import { Table } from "./table";
import { RemoteExclusivity, RemoteVacancy } from
↪  "../../../../../graphql";

export type Teacher = {
  readonly id: string;
  readonly genericID: string;
  name: string;
  url?: string;
};

export type Room = {
  readonly id: string;
  readonly genericID: string;
  name: string;
  olcode?: string;
  description?: string;
};

export type HourMinute = {
  hour: number;
  minute: number;
};
```

```
24
25 export type Schedule = {
26   start: HourMinute;
27   end: HourMinute;
28   dayOfWeek: number;
29   room?: Room | undefined;
30 };
31
32 export type CourseOffer = {
33   id: string;
34   name: string;
35   exclusivity: RemoteExclusivity;
36 };
37
38 type TeamDisciplineOfferOriginal = {
39   id: string;
40   genericID: string;
41   campus: ForeignKey;
42   code: string;
43   name: string;
44 };
45
46 type TeamBase = {
47   readonly id: string;
48   readonly version: string;
49   code: string;
50   schedules: Schedule[];
51   teachers: Teacher[];
52   tables: Table[];
53 };
54
55 type TeamCustom = TeamBase & {
56   custom: true;
57   discipline: TeamDisciplineOfferCustom;
58 };
59
60 type TeamOriginal = TeamBase & {
61   custom: false;
62   discipline: TeamDisciplineOfferOriginal;
```

```
63    vacancies: RemoteVacancy | undefined;
64    course: CourseOffer | undefined;
65  };
66
67  type TeamDisciplineOfferCustom = {
68    id: string;
69    code: string;
70    name: string;
71  };
72
73  export type Team = TeamOriginal | TeamCustom;
74
75  type DisciplineOfferBase = {
76    readonly id: string;
77    readonly version: string;
78    code: string;
79    name: string;
80    teamVersions: { [id: string]: string };
81  };
82
83  type DisciplineOfferCustom = DisciplineOfferBase & {
84    custom: true;
85  };
86
87  type DisciplineOfferOriginal = DisciplineOfferBase & {
88    custom: false;
89    genericID: string;
90    campus: ForeignKey;
91  };
92
93  export type DisciplineOffer = DisciplineOfferOriginal |
    ↪  DisciplineOfferCustom;
94
95  export type DisciplineOfferReference =
96    | {
97        discipline: DisciplineOfferOriginal;
98        deleted: boolean;
99        custom: false;
100       updatedVersion?: string;
```

```
    }
  | {
      discipline: DisciplineOfferCustom;
      deleted: boolean;
      custom: true;
      updatedVersion?: string;
    };
```

Arquivo foreign_key.ts

```
export interface ForeignKey {
  readonly id: string;
  name: string;
}
```

Arquivo index.ts

```
export * from "./discipline_offer";
export * from "./discipline";
export * from "./state";
export * from "./payload";
export * from "./table";
export * from "./plan";
export * from "./selectors";
export * from "./wrapper";
```

Arquivo payload.ts

```
import { HourMinute, Schedule } from "./discipline_offer";
import {
  RemoteLoadDisciplineQuery,
  RemoteLoadDisciplineOfferQuery
} from "../../../../../graphql";

export type PayloadPlanSettings = {
  name: string;
  public: boolean;
};

```

```
export type PayloadCustomPossibility = {
  id: number;
  name: string;
  defaultPossibility: boolean;
};

export type PayloadCreateCustomPossibility = Omit<
  PayloadCustomPossibility,
  "id"
> & {
  copyFrom?: number;
};

export type PayloadCustomDiscipline = {
  id: string;
  code: string;
  name: string;
  team: {
    code: string;
    schedules: Schedule[];
  };
};

export type PayloadCustomDisciplineUpdate = {
  id: string;
  code?: string;
  name?: string;
};

export type PayloadTeamID = {
  offerID: string;
  teamID: string;
};

export type PayloadTeamSelection = PayloadTeamID[];

export type PayloadCustomTeam = PayloadTeamID &
↪  PayloadCustomDiscipline["team"];

```

```
export type PayloadLoadedDisciplineOffer = Exclude<
  RemoteLoadDisciplineOfferQuery["disciplineOffer"],
  undefined
>;

export type PayloadLoadedDiscipline = Exclude<
  RemoteLoadDisciplineQuery["discipline"],
  undefined
>;

export type PayloadPlanID = {
  id: string;
  version: string | null;
};
```

Arquivo plan.ts

```
import { ForeignKey } from "./foreign_key";

export type PlanItem = {
  readonly enabled: boolean;
};
export type PlanItemSelection = PlanItem & {
  readonly selected: boolean;
};

export type PlanTeamSelection = {
  readonly offerID: string;
  readonly teamID: string;
} & PlanItemSelection;

export type PlanPossibilityOffer = {
  color: string;
  version: string;
};

export type PlanDisciplineSelection = {
  readonly disciplineID: string;
} & PlanItem;

```

```
24 export type PlanPossibility = {
25   name: string;
26   offers: { [id: string]: PlanPossibilityOffer };
27   disciplines: { [id: string]: string };
28   selectionDisciplines: PlanDisciplineSelection[];
29   selectionTeams: PlanTeamSelection[];
30 };
31
32 export type PlanVersion = {
33   readonly id: string;
34   savedAt: Date;
35   defaultPossibility: number;
36   possibilities: PlanPossibility[];
37 };
38
39 export type PlanUniversity = {
40   id: string;
41   acronym: string;
42   name: string;
43 };
44
45 export type Plan = {
46   id: string;
47   name: string;
48   lastModified: Date;
49   createdAt: Date;
50   user: ForeignKey;
51   public: boolean;
52   university: PlanUniversity;
53   period: ForeignKey;
54   loadedVersion: PlanVersion;
55   drafts: { [id: string]: PlanVersion };
56   draftsOrder: string[];
57 };
```

Arquivo selectors.ts

```
1 import { PlanItem, PlanItemSelection } from "./plan";
2 import { Discipline } from "./discipline";
3 import { DisciplineOffer, Team } from "./discipline_offer";
```

```
4
5  export type DisciplineSelection = {
6    discipline: Discipline;
7    selection: PlanItem;
8  };
9
10 export type TeamSelection = {
11   team: Team;
12   selection: PlanItemSelection;
13 };
14
15 export type DisciplineOfferSelection = {
16   offer: DisciplineOffer;
17   selection: PlanItemSelection;
18 };
19
20 export type DisciplineOfferTeam = {
21   discipline: DisciplineOffer;
22   team: Team;
23   color: string;
24   selection: PlanItemSelection;
25 };
```

Arquivo state.ts

```
1  import { DisciplineOfferReference, Team } from "./discipline_offer";
2  import { DisciplineReference } from "./discipline";
3  import { Plan } from "./plan";
4
5  export const enum LoadStatus {
6    LOADING,
7    LOADED,
8    ERROR
9  }
10
11 export type OffersMap = {
12   [id: string]: { [version: string]: DisciplineOfferReference };
13 };
14
15 export type DisciplinesMap = {
```

```
  [id: string]: { [version: string]: DisciplineReference };
};

export type SliceState = {
  offers: OffersMap;
  disciplines: DisciplinesMap;
  teams: { [id: string]: { [version: string]: Team } };
  plan: Plan | null;
  status: LoadStatus;
  error?: string;
};

export type FullState = {
  plan: SliceState;
};
```

Arquivo table.ts

```
export type Table = {
  id: string;
  title: string;
  description?: string;
  columns: TableColumn[];
  rows: TableRow[];
};

export type TableColumn = {
  id: string;
  name: string;
};

export type TableRow = {
  row: number;
  column: number;
  value: string;
};
```

Arquivo wrapper.ts

```typescript
import { PayloadAction } from "@reduxjs/toolkit";

export type BaseVersionPossibility = {
  baseVersion: string;
  possibility: number;
};

export type MetaAction = BaseVersionPossibility & {
  targetVersion: string;
};

export type AddVersion<T> = {
  targetVersion: string;
  originalPayload: T;
};

export type MetaVersion<T> = PayloadAction<T, string, MetaAction>;
```

B.3 Extrator de Dados de Habilitações da UFSC

Arquivo build.gradle

```
apply plugin: 'application'

group = 'com.matrufsc2.pdfreader'
version = '1.0-SNAPSHOT'

description = """MatrUFSC2 PDF Reader"""

sourceCompatibility = 1.7
targetCompatibility = 1.7
mainClassName = "com.matrufsc2.pdfreader.Main"


repositories {
     maven { url "http://repo.maven.apache.org/maven2" }
}
dependencies {
    compile group: 'org.apache.pdfbox', name: 'pdfbox', version:'2.0.11'
```

```
18      compile group: 'org.apache.logging.log4j', name: 'log4j-api',
        ↪ version:'2.6.2'
19      compile group: 'commons-codec', name: 'commons-codec', version:
        ↪ '1.11'
20      compile 'com.google.code.gson:gson:2.8.5'
21      compile group: 'com.bpodgursky', name: 'jbool_expressions', version:
        ↪ '1.17'
22      testCompile group: 'junit', name: 'junit', version:'4.12'
23  }
24
25  task fatJar(type: Jar) {
26      manifest {
27          attributes 'Main-Class': mainClassName
28      }
29      baseName = 'pdfreader-all'
30      from { configurations.compile.collect { it.isDirectory() ? it :
        ↪ zipTree(it) } }
31      with jar
32  }
33
34  run {
35      if(project.hasProperty('file')) {
36          args = [project.property('file')]
37      } else {
38          args = ["curriculo.pdf"]
39      }
40  }
```

Arquivo settings.gradle

```
1  rootProject.name = 'pdfreader'
```

B.3.1 Pasta scripts

Arquivo get.js

```
[].slice.call(document.querySelectorAll(".rich-tree-node-text
↪ a")).map((el) => el.href.replace(/;.*\?/, '?')).filter(function(url)
↪ { return url.match("curriculoCurso") }).map((url) => {let curso =
↪ url.match(/curso=(\d+)/)[1]; let curriculo =
↪ url.match(/curriculo=(\d+)/)[1]; return `wget -O
↪ ${curso}-${curriculo}.pdf "${url}"`; }).join("\n")
```

Arquivo open.js

```
[].slice.call(document.querySelectorAll(".rich-tree-node-handle")).forEach(function
↪ {
↪ if(el.querySelector(".rich-tree-node-handleicon-collapsed").style.display
↪ === "none" ) {return} el.click()})
```

B.3.2 Pasta src/main/java/com/matrufsc2/pdfreader

Arquivo CurriculumColumn.java

```java
package com.matrufsc2.pdfreader;

public class CurriculumColumn {
    private String id;
    private String name;

    public CurriculumColumn(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

Arquivo CurriculumCourse.java

```java
package com.matrufsc2.pdfreader;

import java.util.LinkedList;
import java.util.List;

public class CurriculumCourse {
    private String id;
    private String name;
    private String version;
    private List<CurriculumHabilitation> habilitations;

    public CurriculumCourse() {
        this.id = "";
        this.habilitations = new LinkedList<>();
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }


    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
```

```
40      }
41
42      public CurriculumHabilitation getLastHabilitation() {
43          int i = this.habilitations.size()-1;
44          if (this.habilitations.isEmpty() ||
            ↪  this.habilitations.get(i).isFinalized()) {
45              return new CurriculumHabilitation();
46          }
47          return this.habilitations.get(i);
48      }
49      public void addHabilitation(CurriculumHabilitation habilitation) {
50          habilitation.setCourse(this);
51          CurriculumTable detailsTable = habilitation.getTable(0);
52          if (detailsTable.emptyRows()) {
53              habilitation.removeTable(detailsTable);
54          }
55          this.habilitations.add(habilitation);
56      }
57      public boolean hasHabilitationData() {
58          if (this.habilitations.isEmpty()) {
59              return false;
60          }
61          return !getLastHabilitation().getId().isEmpty();
62      }
63
64      public List<CurriculumHabilitation> getHabilitations() {
65          return habilitations;
66      }
67  }
```

Arquivo CurriculumCourseDetails.java

```
1   package com.matrufsc2.pdfreader;
2
3   public class CurriculumCourseDetails {
4       private String id;
5       private String name;
6       private String version;
7
8       public CurriculumCourseDetails(CurriculumCourse course) {
```

```
9           this.id = course.getId();
10          this.name = course.getName();
11          this.version = course.getVersion();
12      }
13
14      public String getId() {
15          return id;
16      }
17
18      public String getName() {
19          return name;
20      }
21
22      public String getVersion() {
23          return version;
24      }
25  }
```

Arquivo CurriculumDiscipline.java

```
1   package com.matrufsc2.pdfreader;
2
3   import java.util.LinkedList;
4   import java.util.List;
5   import java.util.Map;
6
7   /**
8    * Created by fernando on 02/08/16.
9    */
10  public class CurriculumDiscipline {
11      private String id;
12      private String genericId;
13      private String code;
14      private String name;
15      private String description;
16      private String type;
17      private List<CurriculumWorkload> workload;
18      private List<CurriculumTable> tables;
19      private List<CurriculumRelated> related;
20
```

```
21      public CurriculumDiscipline() {
22          this.id = "";
23          this.genericId = "";
24          this.code = "";
25          this.name = "";
26          this.description = "";
27          this.type = "";
28          this.tables = new LinkedList<>();
29          this.related = new LinkedList<>();
30          this.workload = new LinkedList<>();
31      }
32
33      public String getId() {
34          return id;
35      }
36
37      public void setId(String id) {
38          this.id = id;
39      }
40
41      public String getGenericId() {
42          return genericId;
43      }
44
45      public void setGenericId(String genericId) {
46          this.genericId = genericId;
47      }
48
49      public String getCode() {
50          return code;
51      }
52
53      public void setCode(String code) {
54          this.code = code;
55      }
56
57      public String getName() {
58          return name;
59      }
```

```
60
61      public void setName(String name) {
62          this.name = name;
63          if (name.length() > 100) {
64              throw new Error("Discipline name cannot be so long:
                ↪  '"+name+"'");
65          }
66      }
67
68      public String getDescription() {
69          return description;
70      }
71
72      public void setDescription(String description) {
73          this.description = description;
74      }
75
76      public String getType() {
77          return type;
78      }
79
80      public void setType(String type) {
81          this.type = type;
82      }
83
84      private void addHourClass(String type, int hoursClasses) {
85          CurriculumWorkload work = new
            ↪  CurriculumWorkload("total_workload_"+type.toLowerCase().replace("
            ↪  ", "_"), "Número de Horas-Aula", "H/A", hoursClasses);
86          workload.add(work);
87
88      }
89      public void setHoursClasses(int hoursClasses) {
90          addHourClass("UFSC", hoursClasses);
91          if (getType().equalsIgnoreCase("op")) {
92              addHourClass("Optativas Profissionais", hoursClasses);
93          }
94      }
95
```

```
96
97      private CurriculumRelated createRelated(Map<String, String> codes,
        ↪  String type,   List<String> items) {
98          CurriculumRelated related = new CurriculumRelated(type);
99          for (String item: items) {
100             if (item.length() == 7 && item.matches("[A-Z]{3}[0-9]{4}")) {
101                 related.addDiscipline(codes, item);
102             } else {
103                 related.addText(item);
104             }
105         }
106         return related;
107     }
108     public void setClasses(int classes) {
109         CurriculumWorkload work = new CurriculumWorkload("weekly_classes",
            ↪  "Número de Aulas por Semana","Aulas Semanais", classes);
110         workload.add(work);
111     }
112
113     public void addEquivalent(Map<String, String> codes, List<String>
        ↪  equivalent) {
114         this.related.add(createRelated(codes, "equivalent",
            ↪  equivalent));
115     }
116
117
118     public void addPreRequisites(Map<String, String> codes, List<String>
        ↪  preRequisite) {
119         this.related.add(createRelated(codes,"prerequisite",
            ↪  preRequisite));
120     }
121
122
123     public void addSet(Map<String, String> codes, List<String> set) {
124         this.related.add(createRelated(codes,"set",  set));
125     }
126
127     @Override
128     public String toString() {
```

```
129         return "CurriculumDiscipline{" +
130                 "code='" + code + '\'' +
131                 ", name='" + name + '\'' +
132                 ", description='" + description + '\'' +
133                 ", type='" + type + '\'' +
134                 ", workload=" + workload +
135                 ", related=" + related +
136                 '}';
137     }
138 }
```

Arquivo CurriculumHabilitation.java

```
1  package com.matrufsc2.pdfreader;
2
3  import java.util.HashMap;
4  import java.util.LinkedList;
5  import java.util.List;
6  import java.util.Map;
7
8  public class CurriculumHabilitation {
9
10     private String id;
11     private String name;
12     private List<CurriculumLimit> limits;
13     private List<CurriculumStep> steps;
14     private List<CurriculumTable> tables;
15     private CurriculumCourseDetails course;
16     private transient boolean finalized;
17
18     public CurriculumHabilitation() {
19         this.steps = new LinkedList<>();
20         this.limits = new LinkedList<>();
21         this.tables = new LinkedList<>();
22         this.course = null;
23         this.finalized = false;
24         this.name = "";
25         this.id = "";
26     }
27
```

```
28      public void setCourse(CurriculumCourse course) {
29          this.course = new CurriculumCourseDetails(course);
30      }
31
32
33      public void finalizeData() {
34          this.finalized = true;
35      }
36
37      public boolean isFinalized() {
38          return this.finalized;
39      }
40
41
42      public String getId() {
43          return id;
44      }
45
46      public void setId(String id) {
47          this.id = id;
48      }
49
50      public String getName() {
51          return name;
52      }
53
54      public void setName(String name) {
55          this.name = name;
56      }
57
58      public String getObservation() {
59          CurriculumTable table = getTable(0);
60          assert(table.getId().equals("details"));
61          if (table.emptyRows()) {
62              return "";
63          }
64          Map<String, String> row = table.getLastRow();
65          if (!row.get("key").equals("Observação")) {
66              return "";
```

```
67          }
68          return row.get("value");
69      }
70
71      public void setObservation(String observation) {
72          CurriculumTable table = getTable(0);
73          assert(table.getId().equals("details"));
74          Map<String, String> row = null;
75          if (!table.emptyRows()) {
76              row = table.getLastRow();
77          }
78          if (row == null || !row.get("key").equals("Observação")) {
79              row = new HashMap<>();
80              row.put("key", "Observação");
81              table.addRow(row);
82          }
83          assert row.get("key").equals("Observação");
84          row.put("value", observation);
85      }
86
87      public CurriculumCourseDetails getCourse() {
88          return course;
89      }
90
91      public void addStep(CurriculumStep step) {
92          boolean match = steps.isEmpty();
93          if (!match) {
94              CurriculumStep lastStep = steps.get(steps.size()-1);
95              // match = !lastStep.getId().equals(step.getId());
96              // match = match &&
                ↪  !lastStep.getName().equals(step.getName());
97              match = lastStep != step;
98          }
99          if (match) {
100             steps.add(step);
101         }
102     }
103     public void addTable(CurriculumTable table) {
104         tables.add(table);
```

```
105     }
106
107     public void removeTable(CurriculumTable table) {
108         tables.remove(table);
109     }
110
111     public CurriculumTable getTable(int index) {
112         return tables.get(index);
113     }
114
115     public void addLimit(CurriculumLimit limit) {
116         this.limits.add(limit);
117     }
118
119     public List<CurriculumStep> getSteps() {
120         return steps;
121     }
122 }
```

Arquivo CurriculumLimit.java

```
1  package com.matrufsc2.pdfreader;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  public class CurriculumLimit {
7      private String id;
8      private String name;
9      private String context;
10     private String unit;
11     private Map<String, Float> values;
12
13     public CurriculumLimit(String id, String name, String context, String
       ↪ unit) {
14         this.id = id;
15         this.name = name;
16         this.context = context;
17         this.unit = unit;
18         this.values = new HashMap<>();
```

```
19      }
20
21      public void setMin(float min) {
22          this.values.put("min", min);
23      }
24
25      public void setMax(float max) {
26          this.values.put("max", max);
27      }
28
29      public void setIdeal(float ideal) {
30          this.values.put("ideal", ideal);
31      }
32  }
```

Arquivo CurriculumLines.java

```
1   package  com.matrufsc2.pdfreader;
2
3   import java.awt.geom.Line2D;
4   import java.awt.geom.Point2D;
5   import java.util.LinkedList;
6   import java.util.List;
7
8   /**
9    * Created by fernando on 13/08/16.
10   */
11  public class CurriculumLines {
12      private List<Line2D> lines;
13
14
15      public CurriculumLines(List<Line2D> lines) {
16          this.lines = lines;
17      }
18
19      public CurriculumLines() {
20          this(new LinkedList<Line2D>());
21      }
22
23      public boolean isBetween(float y, int i1, int i2) {
```

```
24          return this.isLineAbove(y, i1) && this.isLineBelow(y, i2);
25      }
26
27      public boolean add(Line2D line2D) {
28          if (line2D.getY1() != line2D.getY2()) {
29              // We only add horizontal lines
30              return false;
31          }
32          if (this.lines.isEmpty()) {
33              return this.lines.add(line2D);
34          }
35
36          int i = 0;
37          for (Line2D line: this.lines) {
38              if (line.getY1() == line2D.getY1()) {
39                  return false;
40              }
41              else if (line2D.getY1() < line.getY1()) {
42                  this.lines.add(i, line2D);
43                  return true;
44              }
45              i++;
46          }
47          return this.lines.add(line2D);
48      }
49
50      public String toString() {
51          String result = "";
52          int i = 0;
53          for (Line2D line: this.lines) {
54              result += String.format("Index %d -> Point(%f, %f) to
                ↪  Point(%f, %f)\n", i, line.getX1(), line.getY1(),
                ↪  line.getX2(), line.getY2());
55              i++;
56          }
57          return result;
58      }
59
60      public Line2D get(int index) {
```

```
61          if(index < 0) {
62              index = this.lines.size()+index;
63          }
64          return this.lines.get(index);
65      }
66
67      public boolean isLineBelow(float y, int i ) {
68          return this.get(i).getY1() > y;
69      }
70
71      /**
72       * Returns true if y is smaller than the position of the line at
    ↪  index i. Or, in other words, if the object at
73       * position y is below the line.
74       *
75       * @param y The coordinate Y of the object
76       * @param i The index of the line to compare
77       * @return If the object is below of the line, with the line above
    ↪  the object
78       */
79      public boolean isLineAbove(float y, int i) {
80          return this.get(i).getY1() < y;
81      }
82
83      public boolean add(Point2D p1, Point2D p2) {
84          return this.add(new Line2D.Double(p1.getX(), p1.getY(), p2.getX(),
            ↪  p2.getY()));
85      }
86
87      public int size() {
88          return this.lines.size();
89      }
90
91
92  }
```

Arquivo CurriculumReader.java

```
1  package  com.matrufsc2.pdfreader;
2
```

```
3  import com.bpodgursky.jbool_expressions.And;
4  import com.bpodgursky.jbool_expressions.Expression;
5  import com.bpodgursky.jbool_expressions.Or;
6  import com.bpodgursky.jbool_expressions.Variable;
7  import com.bpodgursky.jbool_expressions.parsers.ExprParser;
8  import com.bpodgursky.jbool_expressions.rules.RuleSet;
9  import org.apache.commons.codec.digest.DigestUtils;
10 import org.apache.logging.log4j.LogManager;
11 import org.apache.logging.log4j.Logger;
12 import org.apache.pdfbox.contentstream.PDFGraphicsStreamEngine;
13 import org.apache.pdfbox.cos.COSName;
14 import org.apache.pdfbox.pdmodel.PDPage;
15 import org.apache.pdfbox.pdmodel.PDPageTree;
16 import org.apache.pdfbox.pdmodel.font.PDFont;
17 import org.apache.pdfbox.pdmodel.graphics.image.PDImage;
18 import org.apache.pdfbox.pdmodel.graphics.state.PDTextState;
19
20 import java.awt.geom.GeneralPath;
21 import java.awt.geom.Point2D;
22 import java.io.IOException;
23 import java.nio.charset.Charset;
24 import java.util.HashMap;
25 import java.util.LinkedList;
26 import java.util.List;
27 import java.util.Map;
28
29 /**
30  * Created by fernando on 02/08/16.
31  */
32 public class CurriculumReader extends PDFGraphicsStreamEngine {
33     private GeneralPath linePath;
34     private int clipWindingRule;
35     private CurriculumLines lines;
36     private CurriculumTexts texts;
37     private float codePosition;
38     private Logger logger = LogManager.getLogger("CurriculumReader");
39     public CurriculumReader() {
40         super(null);
41         this.linePath = new GeneralPath();
```

```
42          this.lines = new CurriculumLines();
43          this.texts = new CurriculumTexts();
44          this.codePosition = -1;
45      }
46
47      @Override
48      protected void showText(byte[] string) throws IOException {
49          PDTextState state = getGraphicsState().getTextState();
50          String str = new String(string, Charset.forName("ISO_8859_1"));
51          if(str.isEmpty()) {
52              return;
53          }
54          float x, y;
55          x = getTextMatrix().getTranslateX();
56          y = getTextMatrix().getTranslateY();
57          PDFont font = state.getFont();
58          this.texts.add(new CurriculumText(x, y, str, font,
            ↪  state.getFontSize()));
59          // We need to update the text matrix correctly, so this method
            ↪  does exactly that..:)
60          super.showText(string);
61      }
62
63      @Override
64      public void processPage(PDPage page) throws IOException {
65          this.linePath = new GeneralPath();
66          this.lines = new CurriculumLines();
67          this.texts = new CurriculumTexts();
68          super.processPage(page);
69      }
70
71      public static String generateHash(String str) {
72          return DigestUtils.sha1Hex(str);
73      }
74
75      public boolean isValidDiscipline(CurriculumState state) {
```

```
76          return state.discipline != null &&
            ↪  !state.discipline.getName().isEmpty() &&
            ↪  !state.discipline.getCode().isEmpty() && state.header.size()
            ↪  > 0 &&
77
                    ↪  !state.discipline.getCode().equals(state.header.get(-1).getStri
78      }
79
80      public String concatenate(String oldString, String newString) {
81          if (oldString == null) {
82              oldString = "";
83          }
84          if (!oldString.isEmpty() && !oldString.endsWith("-")) {
85              oldString = oldString+" ";
86          }
87          return oldString+newString;
88      }
89
90      public String identifyColumn(CurriculumState state, CurriculumText
        ↪  text) throws IOException {
91          for (CurriculumText column: state.header) {
92              if (column.getX() <= text.getX() && column.getX() +
                ↪  column.getWidth() >= text.getX()) {
93                  return column.getString();
94              }
95          }
96          this.logger.warn("Unable to identify column of the text: %s",
            ↪  text.getString());
97          return "";
98      }
99
100     private class CurriculumState {
101         CurriculumStep step;
102         CurriculumDiscipline discipline;
103         Map<String, String> related;
104         float headerPosition;
105         CurriculumTexts header;
106         int startLine;
107         int endLine;
```

```
        CurriculumHabilitation habilitation;
        List<String> preRequisites;
        List<String> equivalents;
        List<String> set;
        boolean nextEquivalent;
        boolean nextPreRequisite;

        CurriculumState(CurriculumHabilitation habilitation) {
            this.habilitation = habilitation;
            this.step = null;
            this.discipline = new CurriculumDiscipline();
            this.related = new HashMap<>();
            this.preRequisites = new LinkedList<>();
            this.equivalents = new LinkedList<>();
            this.header = new CurriculumTexts();
            this.headerPosition = -1;
            this.startLine = -1;
            this.endLine = -1;
            this.set = new LinkedList<>();
            this.nextEquivalent = false;
            this.nextPreRequisite = false;
        }
    }

    private String transformCondition(String condition) {
        // Replace logic tokens of the document with boolean tokens
        String result = "";
        String word = "";
        String quoted = "";
        int parenthesis = 0;
        for(char c: condition.toCharArray()) {
            switch (c) {
                case ')':
                case '(':
                    if (c == '(') {
                        parenthesis++;
                    } else {
                        parenthesis--;
                    }
```

```
147                     if (!word.isEmpty()) {
148                         quoted =  concatenate(quoted, word);
149                         word = "";
150                     }
151                     if (!quoted.isEmpty()) {
152                         result = concatenate(result, "'" + quoted + "'");
153                         quoted = "";
154                     }
155                     result = concatenate(result, ""+c);
156                     break;
157                 case ' ':
158                     boolean special = false;
159                     if (word.equals("eh")) {
160                         word = "&";
161                         special = true;
162                     } else if (word.equals("ou")) {
163                         word = "|";
164                         special = true;
165                     }
166                     if (special) {
167                         if (!quoted.isEmpty()) {
168                             result = concatenate(result, "'" + quoted +
                                ↪  "'");
169                         }
170                         result = concatenate(result, word);
171                         quoted = "";
172                     } else {
173                         quoted =  concatenate(quoted, word);
174                     }
175                     word = "";
176                     break;
177                 default:
178                     word += c;
179             }
180         }
181         if (!word.isEmpty()) {
182             quoted =  concatenate(quoted, word);
183         }
184         if (!quoted.isEmpty()) {
```

```
185             result = concatenate(result, "'" + quoted + "'");
186         }
187         if (parenthesis > 0) {
188             for (int c=0; c<parenthesis; c++) {
189                 result += ")";
190             }
191         }
192         return result;
193     }
194 
195     private String reorderCondition(List<String> condition) {
196         String result = "";
197         for (int i=0, l=condition.size(); i+1 < l; i+= 2) {
198             String prev = condition.get(i);
199             String next = condition.get(i+1);
200             if (prev.length() == 2 && (prev.equals("ou") ||
                ↪  prev.equals("eh"))) {
201                 result = concatenate(result, next.trim());
202                 result = concatenate(result, prev.trim());
203             } else {
204                 result = concatenate(result, prev.trim());
205                 if(!prev.trim().equals(next.trim())) {
206                     result = concatenate(result, next.trim());
207                 }
208             }
209         }
210         if (condition.size()%2 == 1) {
211             result = concatenate(result,
                ↪  condition.get(condition.size()-1));
212         }
213         return result;
214     }
215 
216     private String removeQuote(String quote) {
217         if (quote.startsWith("'")) {
218             quote = quote.substring(1);
219         }
220         if (quote.endsWith("'")) {
221             quote = quote.substring(0, quote.length()-1);
```

```
222         }
223         return quote;
224     }
225 
226     private List<List<String>> fixCondition(List<String> condition) {
227         String result = transformCondition(reorderCondition(condition));
228         // Parse the expression
229         List<List<String>> results = new LinkedList<>();
230         Expression<String> expr = ExprParser.parse(result);
231         expr = RuleSet.simplify(expr);
232         expr = RuleSet.toDNF(expr);
233         // Get all the data! :D
234         if (expr instanceof Variable) {
235             Variable<String> value = (Variable<String>) expr;
236             List<String> list = new LinkedList<>();
237             list.add(removeQuote(value.getValue()));
238             results.add(list);
239         } else if (expr instanceof Or) {
240             Or<String> values = (Or<String>) expr;
241             for (Expression<String> val: values.getChildren()) {
242                 if (val instanceof And) {
243                     fixAndCondition(results, (And<String>) val);
244                 } else {
245                     Variable<String> value = (Variable<String>) val;
246                     List<String> list = new LinkedList<>();
247                     list.add(removeQuote(value.getValue()));
248                     results.add(list);
249                 }
250             }
251         } else {
252             fixAndCondition(results, (And<String>) expr);
253         }
254         return results;
255     }
256 
257     private void fixAndCondition(List<List<String>> results, And<String>
    ↪  expr) {
258         And<String> values = expr;
259         List<String> list = new LinkedList<>();
```

```
260         for (Expression<String> val: values.getChildren()) {
261             Variable<String> value = (Variable<String>) val;
262             list.add(removeQuote(value.getValue()));
263         }
264         results.add(list);
265     }
266
267     private CurriculumState processDiscipline(CurriculumState state,
    ↪  CurriculumText text) throws IOException {
268         state.headerPosition = -1;
269         if (isOutsideDisciplineBounds(state.startLine, state.endLine,
    ↪  text)) {
270             state = this.addDiscipline(state);
271             int it = this.lines.size();
272             state.startLine = identifyDisciplineTopBound(text, it);
273             state.endLine = identifyDisciplineBottomBound(text, it);
274         }
275         if (isDescription(text)) {
276             // Description is full width, so we should read all the text
                ↪  with fontSize=7 to identify it, and we
277             // know that it exists after the code
278
                ↪  state.discipline.setDescription(concatenate(state.discipline.getDescript
                ↪  text.getString()));
279             return state;
280         }
281         if (isName(text)) {
282
                ↪  state.discipline.setName(concatenate(state.discipline.getName(),
                ↪  text.getString()));
283             return state;
284         }
285         if (isCode(text)) {
286             // Code is always at left of the description and the name
287             state.discipline.setCode(text.getString());
288             return state;
289         }
290         switch(identifyColumn(state, text)) {
291             case "H/A":
```

```
292
                    ↪  state.discipline.setHoursClasses(Integer.parseInt(text.getStrin
293                 break;
294             case "Aulas":
295
                    ↪  state.discipline.setClasses(Integer.parseInt(text.getString()))
296                 break;
297             case "Tipo":
298                 String type = text.getString();
299                 switch (type) {
300                     case "Op":
301                         type = "Optativa";
302                         break;
303                     case "Ob":
304                         type = "Obrigatória";
305                         break;
306                     default:
307                         this.logger.warn("Type not recognized:
                            ↪  '"+type+"'");
308                 }
309                 state.discipline.setType(type);
310                 break;
311             case "Pré-Requisito":
312                 state.preRequisites.add(text.getString());
313                 break;
314             case "Equivalentes":
315                 state.equivalents.add(text.getString());
316                 break;
317             case "Conjunto":
318                 state.set.add(text.getString());
319                 break;
320         }
321         return state;
322     }
323
324     private CurriculumState addDiscipline(CurriculumState state) {
325         if (isValidDiscipline(state)) {
326             if (!state.preRequisites.isEmpty()) {
```

```
327                 List<List<String>> lists =
                    ↪ fixCondition(state.preRequisites);
328                 for (List<String> list: lists) {
329                     state.discipline.addPreRequisites(state.related,
                        ↪ list);
330                 }
331                 state.preRequisites = new LinkedList();
332             }
333             if (!state.equivalents.isEmpty()) {
334                 List<List<String>> lists =
                    ↪ fixCondition(state.equivalents);
335                 for (List<String> list: lists) {
336                     state.discipline.addEquivalent(state.related, list);
337                 }
338                 state.equivalents = new LinkedList();
339             }
340             if (!state.set.isEmpty()) {
341                 state.discipline.addSet(state.related, state.set);
342                 state.set = new LinkedList<String>();
343             }
344             String id = state.habilitation.getId()+"-"+
345                     state.habilitation.getCourse().getId()+"-"+
346                     state.step.getId()+"-";
347             if (state.discipline.getCode().length() == 1) {
348                 id += state.discipline.getName() + "-";
349                 id += state.step.getDisciplines().size()+1;
350                 state.discipline.setGenericId("placeholder");
351             } else {
352                 id += state.discipline.getCode();
353
                    ↪ state.discipline.setGenericId(generateHash(state.discipline.getCode
354             }
355             String hash = generateHash(id);
356             state.related.put(state.discipline.getCode(), hash);
357             state.discipline.setId(hash);
358             state.step.add(state.discipline);
359             state.discipline = new CurriculumDiscipline();
360         }
361         return state;
```

```
362     }
363
364     private CurriculumState processTitle(CurriculumState state,
        ↪  CurriculumText text) {
365         state = addDiscipline(state);
366         if (state.step != null) {
367             state.habilitation.addStep(state.step);
368         }
369         state.step = new CurriculumStep(generateHash(text.getString()),
            ↪  text.getString());
370         state.header.clear();
371         state.headerPosition = -2;
372         return state;
373     }
374
375     private CurriculumState processHeader(CurriculumState state,
        ↪  CurriculumText text) {
376         state.headerPosition = text.getY();
377         if (text.getString().equals("Disciplina")) {
378             codePosition = text.getX();
379         }
380         state.header.add(text);
381         return state;
382     }
383
384
385     private class CurriculumDetail {
386         List<String> values;
387         String field;
388         float endField;
389
390         private CurriculumDetail() {
391             this.values = new LinkedList<>();
392             this.field = "";
393             this.endField = -1;
394         }
395     }
396
```

```
397     private void addDetailRow(CurriculumDetail state,
        ↪ CurriculumHabilitation habilitation) {
398         String concat = "";
399         int count = 0;
400         for (String value: state.values) {
401             if (value.contains(":")) {
402                 count++;
403             }
404             concat = concatenate(concat, value);
405         }
406         if (count < state.values.size()) {
407             Map<String, String> row = new HashMap<>();
408             if (state.field.endsWith(":")) {
409                 state.field = state.field.substring(0,
                    ↪ state.field.length()-1);
410             }
411             row.put("key", state.field);
412             row.put("value", concat);
413             habilitation.getTable(0).addRow(row);
414         } else {
415             if (state.field.contains("Período de Conclusão")) {
416                 CurriculumLimit limit = new CurriculumLimit(
                    ↪ "period_limit","Período de Conclusão",
                    ↪ "user","Semestres");
417                 for (String value: state.values) {
418                     int val= Integer.parseInt(value.split(":
                        ↪ ")[1].trim().split(" ")[0].trim());
419                     if (value.contains("Mínimo")) {
420                         limit.setMin(val);
421                     } else if (value.contains("Máximo")) {
422                         limit.setMax(val);
423                     } else if (value.contains("Ideal")) {
424                         limit.setIdeal(val);
425                     }
426                 }
427                 habilitation.addLimit(limit);
428             } else if (state.field.contains("Número de aulas semanais"))
                ↪ {
```

```
429                 CurriculumLimit limit = new
                    ↪ CurriculumLimit("weekly_classes", "Número de Aulas
                    ↪ Semanais", "plan", "Aulas Semanais");
430                 for (String value: state.values) {
431                     int val= Integer.parseInt(value.split(":
                        ↪ ")[1].trim());
432                     if (value.contains("Mínimo")) {
433                         limit.setMin(val);
434                     } else if (value.contains("Máximo")) {
435                         limit.setMax(val);
436                     }
437                 }
438                 habilitation.addLimit(limit);
439             } else if (state.field.contains("Carga Horária Obrigatória"))
                ↪ {
440                 for (String value: state.values) {
441                     String []values = value.split(": ");
442                     String id = values[0].toLowerCase().replace(" ",
                        ↪ "_");
443                     CurriculumLimit limit = new
                        ↪ CurriculumLimit("total_workload_"+id, "Carga
                        ↪ Horária Obrigatória - "+values[0], "user",
                        ↪ "H/A");
444
                        ↪ limit.setMin(Integer.parseInt(values[1].trim().split("
                        ↪ ")[0].trim()));
445                     habilitation.addLimit(limit);
446                 }
447             }
448         }
449         state.values.clear();
450         state.field = "";
451     }
452
453     private CurriculumCourse identifyDetails(PDPage page, CurriculumState
        ↪ globalState, CurriculumCourse course) throws IOException {
454         CurriculumDetail state = new CurriculumDetail();
455         List<CurriculumHabilitation> habilitations =
            ↪ course.getHabilitations();
```

```
456         CurriculumHabilitation habilitation =
            ↪  course.getLastHabilitation();
457         // assert(habilitation == globalState.habilitation);
458         CurriculumTable details = new CurriculumTable("details",
            ↪  "Detalhes", "");
459         details.addColumn(new CurriculumColumn("key", ""));
460         details.addColumn(new CurriculumColumn("value", ""));
461         habilitation.addTable(details);
462         int startAt = -1;
463         for (int i=0, l=this.texts.size()-5; i<l && startAt == -1; i++) {
464             CurriculumText text = this.texts.get(i);
465             if (isTitle(text) && !isOutOfBounds(text)) {
466                 startAt = i;
467             } else if (text.getFontSize() == 9.0) {
468                 String textValue = text.getString();
469                 if (!habilitations.isEmpty() && isObservation(text, i)) {
470                     CurriculumHabilitation oldHabilitation =
                        ↪  habilitations.get(habilitations.size()-1);
471
                        ↪  oldHabilitation.setObservation(concatenate(oldHabilitation.getOb
                        ↪  textValue.trim()));
472                     continue;
473                 }
474                 if (textValue.startsWith("Curso")) {
475                     String[] parts = state.values.get(0).split(" - ", 2);
476                     course.setId(parts[0]);
477                     course.setName(parts[1]);
478                     textValue = "";
479                 } else if (textValue.startsWith("Currículo")) {
480                     course.setVersion(state.values.get(0).substring(0,
                        ↪  4)+"-"+state.values.get(0).substring(4));
481                     course.setId(course.getId()+"-"+course.getVersion());
482                     textValue = "";
483                 } else if (textValue.startsWith("Habilitação: ")) {
484                     String type = textValue.split(": ", 2)[1].trim();
485                     habilitation.setName(type);
486                     textValue = "";
487                 }
488                 state.values.clear();
```

```
489                 state.values.add(textValue);
490             } else if (text.getFontSize() == 8.0) {
491                 String textValue = text.getString();
492                 if (text.getFontName().toLowerCase().endsWith("bold")) {
493                     if (!state.field.isEmpty()) {
494                         this.addDetailRow(state, habilitation);
495                     }
496                     state.field = textValue;
497                     for (int j=i+1; j<l; j++) {
498                         CurriculumText nextField = this.texts.get(j);
499                         if
                            ↪  (nextField.getFontName().toLowerCase().endsWith("bold")
                            ↪  {
500                             state.endField = nextField.getY();
501                             break;
502                         }
503                     }
504                 } else {
505                     if (state.endField < text.getY() ||
                        ↪  state.field.isEmpty()) {
506                         state.values.add(textValue);
507                     } else {
508                         this.addDetailRow(state, habilitation);
509                         state.values.add(textValue);
510                     }
511                 }
512             }
513         }
514         this.addDetailRow(state, habilitation);
515         if (!habilitation.getName().isEmpty()) {
516             globalState.step = null;
517
                ↪  habilitation.setId(generateHash(course.getHabilitations().size()+"-
518             course.addHabilitation(habilitation);
519             globalState.habilitation = habilitation;
520             // assert(globalState.preRequisites.isEmpty());
521             // assert(globalState.equivalents.isEmpty());
522             if (startAt > -1) {
```

```
523                 course = identifyDisciplines(page, globalState, course,
                    ↪  startAt);
524             }
525         }
526         return course;
527     }
528
529     private CurriculumCourse identifyDisciplines(PDPage page,
        ↪  CurriculumState state, CurriculumCourse course) throws
        ↪  IOException {
530         return identifyDisciplines(page, state,  course, 0);
531     }
532
533     private CurriculumCourse identifyDisciplines(PDPage page,
        ↪  CurriculumState state, CurriculumCourse course, int i) throws
        ↪  IOException {
534         CurriculumHabilitation habilitation = state.habilitation;
535         for (int l=this.texts.size()-5; i<l; i++) {
536             CurriculumText text = this.texts.get(i);
537             if (text.getFontName().toLowerCase().endsWith("bold") &&
                ↪  text.getString().toLowerCase().contains("legenda:")) {
538                 habilitation.finalizeData();
539             }
540             if (isObservation(text, i)) {
541
                    ↪  habilitation.setObservation(concatenate(habilitation.getObservation(
                    ↪  text.getString().trim()));
542                 state = addDiscipline(state);
543                 continue;
544             }
545             if (isOutOfBounds(text) && (state.step == null ||
                ↪  state.step.getDisciplines().isEmpty() || i < 6)) {
546                 continue;
547             }
548             if (isTitle(text)) {
549                 state = processTitle(state, text);
550             } else if (isHeader(state.headerPosition, text)) {
551                 state = processHeader(state, text);
552             } else if (state.header.size() > 0) {
```

```
553                 state = processDiscipline(state, text);
554             }
555         }
556         if (state.step != null) {
557             habilitation.addStep(state.step);
558         }
559         return course;
560     }
561 
562     public CurriculumCourse  run(PDPageTree pages, CurriculumCourse
    ↪  course) throws IOException {
563         CurriculumHabilitation habilitation =
        ↪  course.getLastHabilitation();
564         CurriculumState state = new CurriculumState(habilitation);
565         int count = 0;
566         for (PDPage page: pages) {
567             this.processPage(page);
568             if (course.hasHabilitationData()) {
569                 if(count > 0 && state.startLine != -1 && state.endLine !=
                    ↪  -1) {
570                     int size = this.lines.size();
571                     for (int i=0, l=this.texts.size()-5; i<l; i++) {
572                         CurriculumText text = this.texts.get(i);
573                         if (isOutOfBounds(text) && (state.step == null ||
                            ↪  state.step.getDisciplines().isEmpty() || i <
                            ↪  6)) {
574                             continue;
575                         }
576                         if (isCode(text)) {
577                             state.startLine =
                                ↪  identifyDisciplineTopBound(text, size)+1;
578                             state.endLine =
                                ↪  identifyDisciplineBottomBound(text,
                                ↪  size)+1;
579                             break;
580                         }
581                     }
582                 }
583                 course = identifyDisciplines(page, state, course);
```

```
584             } else {
585                 course = identifyDetails(page, state, course);
586             }
587             count++;
588         }
589         return course;
590     }
591 
592     private boolean isOutsideDisciplineBounds(int startLine, int endLine,
    ↪  CurriculumText text) {
593         return !this.lines.isBetween(text.getY(), startLine, endLine);
594     }
595 
596     private int identifyDisciplineBottomBound(CurriculumText text, int
    ↪  it) {
597         for (int il=0; il < it; il++) {
598             if (this.lines.isLineBelow(text.getY(), il)) {
599                 return il;
600             }
601         }
602         this.logger.warn("Unable to identify bottom bound of the
        ↪  discipline");
603         return -1;
604     }
605 
606     private int identifyDisciplineTopBound(CurriculumText text, int it) {
607         for (int il=it-1; il >= 0; il--) {
608             if (this.lines.isLineAbove(text.getY(), il)) {
609                 return il;
610             }
611         }
612         this.logger.warn("Unable to identify bottom bound of the
        ↪  discipline");
613         return -1;
614     }
615 
616     private boolean isOutOfBounds(CurriculumText text) {
617         return this.lines.size() < 4 || !this.lines.isBetween(text.getY(),
        ↪  0, -4);
```

```
618     }
619
620     private boolean isTitle(CurriculumText text) {
621         return text.getFontSize() == 12;
622     }
623
624     private boolean isHeader(float headerPosition, CurriculumText text) {
625         return (headerPosition == -2 || headerPosition == text.getY()) &&
        ↪  text.getFontName().endsWith("Bold");
626     }
627
628     private boolean isCode(CurriculumText text) {
629         return codePosition == text.getX() &&
        ↪  text.getFontName().endsWith("Bold");
630     }
631
632     private boolean isName(CurriculumText text) {
633         return codePosition < text.getX() &&
        ↪  text.getFontName().endsWith("Bold");
634     }
635
636     private boolean isObservation(CurriculumText text, int i) {
637         return text.getFontSize() == 9.0 && i >= 5 && text.getX() == 40.0
        ↪  && !text.getFontName().toLowerCase().endsWith("bold");
638     }
639
640     private boolean isDescription(CurriculumText text) {
641         return text.getFontSize() == 7 && codePosition < text.getX();
642     }
643
644     @Override
645     public void appendRectangle(Point2D p0, Point2D p1, Point2D p2,
        ↪  Point2D p3) throws IOException {
646         // to ensure that the path is created in the right direction, we
        ↪  have to create
647         // it by combining single lines instead of creating a simple
        ↪  rectangle
648         linePath.moveTo((float) p0.getX(), (float) p0.getY());
649         this.lineTo((float) p1.getX(), (float) p1.getY());
```

```
650             this.lineTo((float) p2.getX(), (float) p2.getY());
651             this.lineTo((float) p3.getX(), (float) p3.getY());
652 
653             // close the subpath instead of adding the last line so that a
                ↪  possible set line
654             // cap style isn't taken into account at the "beginning" of the
                ↪  rectangle
655             // linePath.closePath();
656             // We need to use this so we can track the horizontal lines of
                ↪  the PDF
657             this.lineTo((float) p0.getX(), (float) p0.getY());
658         }
659 
660         @Override
661         public void drawImage(PDImage pdImage) {
662         }
663 
664         @Override
665         public void clip(int windingRule) {
666             clipWindingRule = windingRule;
667         }
668 
669         @Override
670         public void moveTo(float x, float y) {
671             linePath.moveTo(x, y);
672         }
673 
674         @Override
675         public void lineTo(float x, float y) throws IOException {
676             Point2D oldPoint, newPoint;
677             oldPoint = this.getCurrentPoint();
678             newPoint = new Point2D.Float(x, y);
679             this.lines.add(oldPoint, newPoint);
680             linePath.lineTo(x, y);
681         }
682 
683         @Override
684         public void curveTo(float x1, float y1, float x2, float y2, float x3,
                ↪  float y3) {
```

```
685             linePath.curveTo(x1, y1, x2, y2, x3, y3);
686         }
687 
688         @Override
689         public Point2D getCurrentPoint() {
690             Point2D result = linePath.getCurrentPoint();
691             if (result == null) {
692                 result = new Point2D.Float(0, 0);
693             }
694             return result;
695         }
696 
697         @Override
698         public void closePath() {
699             linePath.closePath();
700         }
701 
702         @Override
703         public void endPath() {
704             if (clipWindingRule != -1)
705             {
706                 linePath.setWindingRule(clipWindingRule);
707                 getGraphicsState().intersectClippingPath(linePath);
708                 clipWindingRule = -1;
709             }
710             linePath.reset();
711         }
712 
713         @Override
714         public void strokePath() {
715             linePath.reset();
716         }
717 
718         @Override
719         public void fillPath(int windingRule) {
720             linePath.reset();
721 
722         }
723 
```

```
724     @Override
725     public void fillAndStrokePath(int windingRule) {
726         linePath.reset();
727 
728     }
729 
730     @Override
731     public void shadingFill(COSName shadingName) {
732 
733     }
734 }
```

Arquivo CurriculumRelated.java

```
1  package com.matrufsc2.pdfreader;
2  
3  import org.apache.commons.codec.digest.DigestUtils;
4  
5  import java.util.LinkedList;
6  import java.util.List;
7  import java.util.Map;
8  
9  public class CurriculumRelated {
10     private class Item {
11         private String type;
12         private String value;
13         private String description;
14 
15         public Item(String type, String value, String description) {
16             this.value = value;
17             this.type = type;
18             this.description = description;
19         }
20     }
21     private String name;
22     private String type;
23     private List<Item> items;
24 
25     public CurriculumRelated(String type) {
26         if (type.equals("prerequisite")) {
```

```
27              this.name = "Pré-Requisitos";
28          } else if (type.equals("equivalent")) {
29              this.name = "Equivalentes";
30          } else if (type.equals("set")) {
31              this.name = "Conjunto";
32          }
33          this.type = type;
34          this.items = new LinkedList<>();
35      }
36
37      public void addDiscipline(Map<String, String> codes, String code) {
38          Item result;
39          if (!codes.containsKey(code)) {
40              result = new Item("discipline_generic_id",
                ↪  DigestUtils.sha1Hex(code), code);
41          } else {
42              result = new Item("discipline_id", codes.get(code), code);
43          }
44          this.items.add(result);
45      }
46
47      public void addText(String value) {
48          this.items.add(new Item("text", value, value));
49      }
50  }
```

Arquivo CurriculumStep.java

```
1   package  com.matrufsc2.pdfreader;
2
3   import java.util.LinkedList;
4   import java.util.List;
5
6   public class CurriculumStep {
7       private String id;
8       private List<CurriculumDiscipline> disciplines;
9       private String name;
10
11      public CurriculumStep(String id, String name) {
12          this.id = id;
```

```
13          this.name = name;
14          this.disciplines = new LinkedList<>();
15      }
16
17      public String getId() {
18          return id;
19      }
20
21      public String getName() {
22          return name;
23      }
24
25      public void add(CurriculumDiscipline discipline) {
26          this.disciplines.add(discipline);
27      }
28
29      public List<CurriculumDiscipline> getDisciplines() {
30          return this.disciplines;
31      }
32
33      @Override
34      public String toString() {
35          return "CurriculumStep{" +
36                  "disciplines=" + disciplines +
37                  ", name='" + name + '\'' +
38                  '}';
39      }
40  }
```

Arquivo CurriculumTable.java

```
1   package com.matrufsc2.pdfreader;
2
3   import java.util.LinkedList;
4   import java.util.List;
5   import java.util.Map;
6
7   public class CurriculumTable {
8       private String id;
9       private String title;
```

```
10      private String description;
11      private List<CurriculumColumn> columns;
12      private List<Map<String, String>> rows;
13
14      public CurriculumTable(String id, String title, String description) {
15          this.id = id;
16          this.title = title;
17          this.description = description;
18          this.columns = new LinkedList<>();
19          this.rows = new LinkedList<>();
20      }
21
22      public String getId() {
23          return id;
24      }
25
26      public String getTitle() {
27          return title;
28      }
29
30      public String getDescription() {
31          return description;
32      }
33
34      public boolean emptyRows() {
35          return this.rows.isEmpty();
36      }
37      public Map<String, String> getLastRow() {
38          return this.rows.get(this.rows.size()-1);
39      }
40
41      public void addColumn(CurriculumColumn column) {
42          this.columns.add(column);
43      }
44
45      public void addRow(Map<String, String> row) {
46          this.rows.add(row);
47      }
48  }
```

Arquivo CurriculumText.java

```
1  package  com.matrufsc2.pdfreader;
2
3  import org.apache.pdfbox.pdmodel.font.PDFont;
4  import org.apache.pdfbox.pdmodel.font.PDFontDescriptor;
5
6  import java.io.IOException;
7
8  /**
9   * Created by fernando on 14/08/16.
10  */
11 public class CurriculumText {
12     protected float x, y, fontSize;
13     protected String string;
14     protected PDFont font;
15
16     public CurriculumText(float x, float y, String string, PDFont font,
       ↪  float fontSize) {
17         this.x = x;
18         this.y = y;
19         this.string = string;
20         this.font = font;
21         this.fontSize = fontSize;
22     }
23
24     public float getX() {
25         return x;
26     }
27
28     public float getY() {
29         return y;
30     }
31
32     public String getString() {
33         return string;
34     }
35
36     public PDFont getFont() {
```

```
37          return font;
38      }
39
40      public PDFontDescriptor getFontDescriptor() {
41          return this.getFont().getFontDescriptor();
42      }
43
44      public String getFontFamily() {
45          return this.getFontDescriptor().getFontFamily();
46      }
47
48      public String getFontName() {
49          return this.getFontDescriptor().getFontName();
50      }
51
52      public float getFontSize() {
53          return fontSize;
54      }
55
56      public float getWidth() throws IOException {
57          return this.getFont().getStringWidth(this.getString());
58      }
59
60      @Override
61      public String toString() {
62          return "CurriculumText{" +
63                  "x=" + x +
64                  ", y=" + y +
65                  ", fontSize=" + fontSize +
66                  ", string='" + string + '\'' +
67                  ", font=" + font +
68                  '}';
69      }
70  }
```

Arquivo CurriculumTexts.java

```
1   package  com.matrufsc2.pdfreader;
2
3   import java.util.Iterator;
```

```
4  import java.util.LinkedList;
5  import java.util.List;
6
7  /**
8   * Created by fernando on 14/08/16.
9   */
10 public class CurriculumTexts implements Iterable<CurriculumText> {
11     protected List<CurriculumText> texts;
12
13     public CurriculumTexts(List<CurriculumText> texts) {
14         this.texts = texts;
15     }
16
17     public CurriculumTexts() {
18         this(new LinkedList<CurriculumText>());
19     }
20
21     public boolean isBetween(float y, int i1, int i2) {
22         return this.isAbove(y, i1) && this.isBelow(y, i2);
23     }
24
25     public void clear() {
26         texts.clear();
27     }
28     public boolean add(CurriculumText text) {
29         if (this.texts.isEmpty()) {
30             return this.texts.add(text);
31         }
32
33         int i = 0;
34         for (CurriculumText textList: this.texts) {
35             if (textList.getY() == text.getY()) {
36                 // We do not allow duplicates
37                 if (text.getX() > textList.getX()) {
38                     this.texts.add(i, text);
39                     return true;
40                 }
41             }
42             else if (text.getY() > textList.getY()) {
```

```
43                  this.texts.add(i, text);
44                  return true;
45              }
46              i++;
47          }
48          return this.texts.add(text);
49      }
50
51
52      public String toString() {
53          String result = "";
54          int i = 0;
55          for (CurriculumText text: this.texts) {
56              result += String.format("Index %d -> Point(%f, %f) - %s\n", i,
                ↪  text.getX(), text.getY(), text.getString());
57              i++;
58          }
59          return result;
60      }
61
62      @Override
63      public Iterator<CurriculumText> iterator() {
64          return this.texts.iterator();
65      }
66
67      public CurriculumText get(int index) {
68          if(index < 0) {
69              index = this.size()+index;
70          }
71          return this.texts.get(index);
72      }
73
74      public boolean isBelow(float y, int i ) {
75          return this.get(i).getY() > y;
76      }
77
78      public boolean isAbove(float y, int i) {
79          return this.get(i).getY() < y;
80      }
```

```
81
82      public int size() {
83          return this.texts.size();
84      }
85  }
```

Arquivo CurriculumWorkload.java

```
1   package com.matrufsc2.pdfreader;
2
3   public class CurriculumWorkload {
4       private String id;
5       private String name;
6       private String unit;
7       private float value;
8
9       public CurriculumWorkload(String id, String name, String unit, float
        ↪  value) {
10          this.id = id;
11          this.name = name;
12          this.unit = unit;
13          this.value = value;
14      }
15  }
```

Arquivo Main.java

```
1   package com.matrufsc2.pdfreader;
2
3   import com.google.gson.FieldNamingPolicy;
4   import com.google.gson.Gson;
5   import com.google.gson.GsonBuilder;
6   import org.apache.pdfbox.cos.COSObject;
7   import org.apache.pdfbox.io.MemoryUsageSetting;
8   import org.apache.pdfbox.pdmodel.DefaultResourceCache;
9   import org.apache.pdfbox.pdmodel.PDDocument;
10  import org.apache.pdfbox.pdmodel.PDPage;
11  import org.apache.pdfbox.pdmodel.PDPageTree;
12  import org.apache.pdfbox.pdmodel.graphics.PDXObject;
```

```
13
14 import java.io.*;
15 import java.util.ArrayList;
16 import java.util.List;
17
18 public class Main {
19     private static class NullCache extends DefaultResourceCache
20     {
21         @Override
22         public void put(COSObject indirect, PDXObject xobject) throws
           ↪ IOException
23         {
24         }
25     }
26
27     private static StringBuilder parsePDF(Gson encoder, boolean first,
       ↪ File file, StringBuilder resultado) {
28
29         CurriculumReader reader;
30         PDDocument doc = null;
31         try {
32             doc = PDDocument.load(file,
               ↪ MemoryUsageSetting.setupTempFileOnly());
33             doc.setResourceCache(new NullCache());
34             PDPageTree pages = doc.getPages();
35             CurriculumCourse course = new CurriculumCourse();
36             reader = new CurriculumReader();
37             course = reader.run(pages, course);
38             for (CurriculumHabilitation hab: course.getHabilitations()) {
39                 if (!first) {
40                     resultado.append(",");
41                 }
42                 first = false;
43                 resultado.append(encoder.toJson(hab));
44             }
45         } catch (Exception e) {
46             System.err.printf("Error while processing file '%s'\n",
               ↪ file);
47             e.printStackTrace();
```

```
48          }
49          finally {
50              if (doc != null) {
51                  try {
52                      doc.close();
53                  } catch (IOException e) {
54                      System.err.printf("Error while processing file
                        ↪  '%s'\n", file);
55                      e.printStackTrace();
56                  }
57              }
58          }
59          return resultado;
60      }
61      public static void main(String[] args) {
62          Gson encoder = new
            ↪  GsonBuilder().setFieldNamingPolicy(FieldNamingPolicy.LOWER_CASE_WITH_UNDERSC
63
64          File folder = new File(args[0]);
65          File []files;
66          if (folder.isFile()) {
67              files = new File[]{folder};
68          } else {
69              files = folder.listFiles();
70          }
71          StringBuilder resultado = new StringBuilder();
72          for (int i = 0, l = files.length; i < l; i++) {
73              resultado = parsePDF(encoder, i == 0, files[i], resultado);
74              System.gc();
75          }
76          resultado.append("]");
77          System.out.println(resultado.toString());
78      }
79
80  }
```