UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMATICA E ESTATISTICA

Arthur Henrique Della Fraga e Nathan Junior Molinari

PORTAL DO COLABORADOR MEUBRIDGE: MÓDULO DE CONTROLE DE HORAS E FÉRIAS

Florianópolis

2018

Arthur Henrique Della Fraga e Nathan Junior Molinari

PORTAL DO COLABORADOR MEUBRIDGE: MÓDULO DE CONTROLE DE HORAS E FÉRIAS

Trabalho de Conclusão de Curso submetido ao Departamento de Informática e Estatística para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Raul Sidnei Wazlawick

Florianópolis

2018

Arthur Henrique Della Fraga e Nathan Junior Molinari

PORTAL DO COLABORADOR MEUBRIDGE: MÓDULO DE CONTROLE DE HORAS E FÉRIAS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do Título de "Bacharel em Ciências da Computação" e aprovado em sua forma final pela Departamento de Informática e Estatística.

Florianópolis, 23 de Novembro 2018.

Prof. Dr. Eng. Rafael Luiz Cancian
Coordenador do Curso

Banca Examinadora:

Prof. Raul Sidnei Wazlawick
Orientador

Prof. Antônio Carlos Mariani

Bel. Gabriel Holdener Geraldeli

RESUMO

O trabalho consiste em avaliar soluções de software e propor uma ferramenta web open source para auxiliar atividades administrativas de uma instituição, gerenciando a carga horária, período de atividade de trabalho e férias de seus colaboradores.
O sistema permite a um usuário, participante de uma organização, determinar seu horário de trabalho semanal e, a partir do registro diário de suas entradas e saídas, acompanhar quanto do esperado este já trabalhou.
Também é possível ao participante verificar quanto de férias tem direito, bem como determinar seu período de utilização. Ou ainda, acompanhar feriados e registrar motivos de ausência ou abono de um dia de trabalho.
O projeto visa modelar e implementar uma solução de software para determinadas necessidades administrativas identificadas no laboratório Bridge, desenvolvendo uma ferramenta de utilização pública que servirá de base para futuros trabalhos acadêmicos guiados por dar continuidade ao sistema em contribuição a outras vivências organizacionais.

Palavras-chave: Gestão de pessoas, Banco de horas, Férias, Aplicação web, API REST.

ABSTRACT

The project's objective is to develop an open source web platform to help administrate activities of an institution, managing the workload, daily work activity period and vacations of its collaborators.
The system enables for an user, participant in an organization, to determine his weekly work schedule and, from the daily register, track how much time is missing compared to the expected.
It's also possible for the user to check how many vacation days he has accumulated, as well as determine its period of use, or to record reasons for missing a work day.
The project intends to model and implement a solution for some of the administrative needs identified in the Bridge laboratory, structuring a platform that works as a base to be expanded through future projects developed in the organization or throught other academic projects.

Keywords: People management. Time control, Vacation, Web application, REST API.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Organizações são formadas por pessoas e é a partir do tempo investido por elas que se realizam as atividades as quais essa se propõe. Visualizando isso, dimensionar a disponibilidade de força de trabalho no tempo torna-se tarefa indispensável na execução de qualquer projeto.

Para que esse planejamento ocorra com mais precisão é prático que haja um método no qual os envolvidos esclareçam qual volume de tempo dedicarão às demandas da organização, e muitas vezes em quais períodos pretendem realizar suas responsabilidades.

A partir disso, é importante que este processo sirva a todos, seja escalável, conferível e, lógico, viável economicamente. Isso é, um método em que o planejamento citado possa ser realizado para um grande número de participantes, bem como verificado o seu cumprimento e mesmo assim não demande grandes investimentos.

É então, com o objetivo de cooperar com essa intrínseca necessidade do gerenciamento de um projeto, que este trabalho propõe a concepção e desenvolvimento de um sistema computacional open source que auxilie certas atividades administrativas da gestão de pessoas em uma instituição.

A intenção é estruturar o armazenamento de informações sobre os envolvidos da organização de modo a permitir que diversos softwares possam consumir, operar e contribuir com esse repositório central. E, em complemento, desenvolver uma aplicação que, por hora, gerencie a jornada de atividade semanal dos participantes, seu período diário de trabalho, seu planejamento de férias e eventuais ausências justificadas.

A utilidade de um instrumento como esse é percebida tomando como referência a vivência organizacional dos colaboradores, bolsistas ou CLTs, do laboratório de desenvolvimento tecnológico da UFSC Bridge. O projeto visa avaliar e propor soluções de software que satisfaçam os requisitos organizacionais levantados, implementando uma ferramenta que proporcione evolução ao atual método de gestão de banco de horas empregado na organização. Mais que isso: Construir uma base ontológica sobre seus participantes e com isso fundar uma arquitetura de dados expansível que sirva de esqueleto para a implementação de futuras funcionalidades complementares ao gerenciamento de carga horária, aqui proposto.

Nos próximos capítulos esclareceremos os objetivos do trabalho, justificaremos sua importância, discutiremos softwares semelhantes, descreveremos o processo adotado para seu desenvolvimento, defi-

niremos alguns conceitos e tecnologias utilizadas, listaremos requisitos do sistema.

2 OBJETIVOS

2.1 GERAIS

A finalidade desse trabalho é desenvolver um sistema computacional que gerencie o período de atividade de trabalho dos envolvidos de uma organização, permitindo-os discriminar seu ciclo de horários semanal, registrar tempo diário de dedicação, planejar retirada de férias e declarar motivos de faltas.

Se deseja estabelecer um ambiente onde os usuários possam confrontar o volume de tempo investido na instituição com aquele que haviam proposto fazer. Um espaço onde se possa consultar o horário de trabalho dos demais colaboradores, ou ainda acompanhar o calendário de eventos e dias de expediente da organização. Por extensão, uma ferramenta que descreva ao interessado sua disponibilidade de férias e permita-o definir quando as gozará.

2.2 ESPECÍFICOS

De maneira mais detalhada, especificaremos melhor os objetivos desse projeto. Vale citar, quanto ao desenvolvimento da aplicação em si, a disponibilização de algumas funcionalidades. Dentre elas:

- Configuração de carga horária semanal;
- Registro de jornada diária de trabalho;
- Gestão de férias;
- Acesso ao calendário de eventos da organização;

Ainda quanto a sua implementação, a ferramenta será estruturada computacionalmente visando sua expansão. Seu acesso simplificado com e-mail do Google e concordância com a identidade visual do laboratório também serão considerados.

Por fim, também consta como propósito, avaliar as melhorias proporcionadas pelo produto final, em relação ao método vigente, quanto aos pontos:

- Funcionalidades disponíveis;
- Satisfação do usuário;

- Usabilidade.

As medidas alcançadas e opiniões coletadas quanto ao progresso obtido serão expostas no próprio corpo do trabalho.

3 JUSTIFICATIVA

Dentre as políticas organizacionais do laboratório Bridge, consta a flexibilidade de horários, na qual colaboradores podem organizar seu período de dedicação de acordo com demais compromissos acadêmicos e pessoais que possuam. Mais que isso, possuem a liberdade de se ausentarem conforme necessitem, compensando aquelas horas de trabalho em outro momento.

E foi buscando prover um método de acompanhamento dessas questões que, ainda em 2014, fora desenvolvida dentro da instituição uma planilha que estruturava o registro dos períodos diários de atividade de trabalho do colaborador. Em 2015 e 2016, outras duas versões da ferramenta foram elaboradas, trazendo melhorias na organização, interface, configuração, performance, bem como novas funções.

Com a consolidação dessa solução como estratégia gerencial, cada vez mais era desejado a correção de imperfeições ou a implementação de novas funcionalidades, entretanto as características da tecnologia utilizada e o esforço requerido para aplicar essas mudanças tornaram-se limitantes para essas evoluções. Sendo assim, a versão mais atual da planilha mantém-se em utilização sem transformações há quase 3 anos.

Nesse prazo se percebeu a utilidade que outros instrumentos computacionais poderiam proporcionar à administração do laboratório, e foi tendo isso em vista que se decidiu pela implementação de um sistema centralizador de serviços que apoie as tarefas relacionadas à gestão de pessoas e de processos administrativos empregados na organização.

Logo, o software produzido a partir deste trabalho será pioneiro neste processo, projetando, desenvolvendo e dispondo uma arquitetura de dados, modelo computacional, API web e um software cliente que servirão de ponto de partida para implementação de demais de serviços que interessem à vivência organizacional do laboratório. Em complemento, trará os serviços geridos pela planilha de horas citada para um sistema de informação, promovendo maior eficiência e trazendo novas funções.

4 METODOLOGIA

Para alcançar as metas firmadas se estabeleceu um processo que parte da atual convenção de gerenciamento empregada pela organização, esclarece seus preceitos e intuitos, estuda propostas, constitui uma solução e a aplica, avaliando por fim os benefícios adquiridos.

De maneira mais estruturada, o método de trabalho correrá pelas seguintes etapas:

1. Conhecer a atual ferramenta de gerenciamento da jornada de trabalho e controle de férias utilizado no Laboratório Bridge e para esse identificar e descrever as funcionalidades disponíveis.

2. Reconhecer, com a gerência da organização e usuários da ferramenta, quais os pontos críticos essa apresenta e que outros serviços poderiam expandir sua contribuição nas demandas administrativas relacionadas.

3. Pesquisar e avaliar softwares existentes que sirvam às exigências levantadas.

4. Propor uma ferramenta computacional que proporcione evolução ao modelo avaliado no que diz respeito às limitações registradas e utilidades desejáveis.

5. Documentar formalmente o funcionamento esperado da aplicação, entregando material de apoio ao desenvolvimento e teste do software.

6. Modelar uma arquitetura de dados que estruture informações sobre usuários participantes de uma fundação visando que tal repositório sirva de âmago para o implementação de diversos serviços relacionados à gestão dessas pessoas.

7. Desenvolver uma API web para comunicação de distintos sistemas com a base de dados modelada, permitindo leitura e escrita de informações de maneira controlada.

8. Instanciar uma execução desse repositório.

9. Implementar a ferramenta proposta, de maneira que essa interaja com a base de dados em execução, servindo assim como exemplo prático da utilização da API elaborada.

10. Disponibilizar uma versão do sistema em produção.

11. Aplicar o software desenvolvido no cotidiano do laboratório, substituindo a ferramenta atual, e então avaliar, através de um questionário com os usuários, se este proporciona as melhorias que se propunha.

Os artefatos resultantes das etapas do processo estão expostos nas discussões e evidências apresentadas ao longo do texto que segue.

5 TRABALHOS CORRELATOS

Para a avaliação de outros mecanismos que disponham serviços semelhantes aos que propomos, restringimos nossa comparação à sistemas web de controle de horas que fossem gratuitos e / ou open source, ambas características empregadas em nossa proposta.

A gratuidade se fez necessária visto que os recursos investidos no laboratório tem como finalidade exclusiva financiar os projetos aos quais se destinam, não restando assim quantia para desenvolvimentos da própria instituição. Já a abertura do código tem como objetivo deixar essa contribuição disponível à comunidade computacional, permitindo que qualquer instituição possa se utilizar da ferramenta e ainda reforçando a intenção de expansão da plataforma, possibilitando a incorporação de futuros serviços.

5.1 MEU CONTROLE DE PONTO

A aplicação Meu controle de ponto (www.meucontroledeponto.com.br) provê um ambiente para gestão pessoal de horas investidas em atividades, não se restringindo a uma visão empregatícia desse cenário, abordando, por exemplo, planejamento de estudos ou uma perspectiva para trabalhadores autônomos.

Essa também possibilita a geração de relatórios sobre as informações inseridas, disponibiliza a gestão de banco de horas para períodos determinados e ainda alerta o usuário a respeito de certas infrações da legislação trabalhista.

5.2 PSTIME

A plataforma gratuita PSTime (www.pstime.com.br) permite a usuários o registro e acompanhamento das horas dedicadas à empresa. Trazendo uma abordagem mais direcionada ao gerenciamento de projetos, essa possibilita aos participantes discriminarem quanto do período trabalhado destinaram a cada uma de suas atividades. Dispõe também de mecanismos para registros de gastos relacionados às tarefas exercidas pelo envolvido.

E ainda vai mais longe, permitindo a administradores cadastrar projetos, organizar suas equipes e descrever as atividades que nesse

serão exercidas. Oferece inclusive ambiente para consulta do volume de tempo e de gastos investido nas atividades de seus projetos.

Além disso, a aplicação conta com uma vasta API através da qual seus clientes podem manejar todas as funcionalidades mencionadas, integrando, assim, o uso do PSTime à outras ferramentas já empregadas nos processos da organização.

5.3 PRIMAERP

A ferramenta PrimaERP (www.primaerp.com) possui, dentre outros módulos, um dedicado a contabilização de tempo de trabalho. Com uma interface simples e prática é possível registrar início e fim de expediente com apenas um clique, marcando também pausas para café, almoço ou outras atividades.

Os registros diários descrevem qual horário o funcionário chegou na empresa e que horas esse foi embora, discriminando tabular e graficamente qual percentual de tempo trabalhou ou esteve em intervalos. Persistindo os dias trabalhados para consulta futura.

5.4 RESUMO COMPARATIVO

Já à primeira avaliação é fácil identificar que as ferramentas comparadas possuem objetivos distintos dos que buscamos. Seja por possuir uma abordagem mais projeto-gerencial (como é o caso do PSTime), incorporando descrição de gastos ou tempos dedicado a atividades e, por consequência, exigindo configuração prévia robusta, além de poluir sua interface com funções indesejadas. Ou por ter proposta demasiadamente simplória (como o PrimaERP) ou individual (como o Meu controle de ponto), não compreendendo o que é desejado.

O descarte das plataformas se consolida quando observamos a ausência das principais funcionalidades cobiçadas. Isso é, nenhuma delas contabiliza banco de horas, controla férias ou armazena horário semanal dos usuários, resumindo-se apenas a registrar suas entradas e saídas para simples contabilização diária de período de atividade.

Vale também pontuar que todas as aplicações (PSTime, Meu Controle de ponto e PrimaERP) não suprem a vontade de um repositório que centralize informações acerca dos colaboradores e da instituição, o que impossibilita o desenvolvimento de futuras funcionalidades fora do âmbito do controle de horas e férias.

Tabela 1 – Comparação entre as funcionalidades dos sistemas correlatos

	Bridge	Meu controle de ponto	PSTime	PrimaERP
Cadastro de jornada semanal	Sim	Sem discriminação de horários	Não	Não
Registro diário de horas trabalhadas	Sim	Sim	Sim	Sim
Discriminação de horas investidas por atividade	Não	Não	Sim	Sim
Banco de horas	Sim	Sim	Não	Não
Controle de férias	Sim	Não	Não	Não
Registro de gastos	Não	Não	Sim	Sim
Registro de ausências justificadas	Sim	Sim	Não	Não
Acompanhamento do calendário de atividades da organização	Sim	Não	Não	Não
API disponível	Sim	Não	Sim	Sim
Número de usuários	Ilimitado	Contas individuais independentes	Ilimitado	Apenas 3

6 FUNDAMENTAÇÃO TEÓRICA

6.1 BANCO DE HORAS

Nos modelos clássicos de relação de trabalho o período de atividade dos empregados sempre foi muito bem definido, tendo por estipulado seu horário de entrada, de saída e seus intervalos. Entretanto, com o passar dos anos, a rigidez dessas e outras estipulações tem dado espaço a um vínculo empregatício com enfoque na cooperação, estabelecendo relações de confiança e delegando autonomia aos envolvidos, relação de trabalho que (OCUPACIONAL, 1997) indica favorecer a produtividade, criatividade e trabalho em equipe, aumentando a auto-estima dos participantes e contribuindo para melhorar sua qualidade de vida e satisfação no trabalho.

Para além de inúmeras iniciativas que busquem esses benefícios, uma política de banco de horas atua diretamente na expectativa sobre período de atividade de trabalho do empregado. Essa abordagem prevê que o envolvido tenha a disponibilidade para participar de eventuais compromissos pessoais sem comprometer o volume de trabalho que dedica às suas funções, adaptando seus horários para compreender aquela circunstância.

Essa prática está presente sobretudo no mercado de desenvolvimento de software, que vezes ainda estabelece modelos de home office, permitindo aos seus funcionários também decidirem de onde desenvolverão suas atividades. Tais modelos de relação trabalhista são previstos em lei pelo (Governo Federal Brasileiro, 1943), qual determina direitos e deveres a favor de um regime de compensação de horas de trabalho. A abordagem é comum em propostas de bolsas e estágios oferecidas por laboratórios da UFSC - a citar, o LabSec e o Lisha, do INE - bem como no Laboratório Bridge, realidade decisiva para incorporação dessa gestão no escopo de desenvolvimento do software proposto nesse trabalho.

6.2 API REST

Application Program Interface (API), no universo da World Wide Web (WWW), pode ser vista como uma forma flexível de acesso a um banco de dados que proporciona interoperabilidade entre aplicações (Open Api Initiative, 2017). E, dentre diferentes maneiras para imple-

mentação de uma, o paradigma REST tem sido adotado no desenvolvimento de web services modernos (YATES et al., 2014) por inúmeros benefícios associados.

O termo REST (REpresentational State Transfer) foi criado por Roy Fielding (FIELDING, 2000) que se baseou no funcionamento da arquitetura Web para criar princípios de design visando "minimizar a latência e as comunicações de rede e, ao mesmo tempo, maximizar a independência e a escalabilidade das implementações de componentes"(LI; CHOU, 2011).

Para além disso, de acordo com (PATNI, 2017), uma API implementada sob os princípios REST irá aprimorar: performance, escalabilidade e modificação dos componentes, portabilidade, confiabilidade e visibilidade. Razões pelas quais tomamos essa opção.

Por se tratar de uma arquitetura, REST é independente de linguagem de programação ou qualquer tecnologia específica, assim como não define o tipo dos dados a serem operados - permitindo, portanto, que esses sejam consumidos por qualquer tipo de aplicação, seja essa web, desktop ou móvel. O que não trará restrições sobre as possibilidades de desenvolvimento de futuras funcionalidades.

O conceito REST, na verdade, diz respeito a uma série de princípios adotados para a interação entre sistemas. Dentre eles o de não gerenciamento do estado da conexão entre as partes. Ou o uso do protocolo HTTP para acesso e manipulação de representações textuais dos recursos da aplicação. Aqui, recursos podem ser textos, mídias, áudios, vídeos, não há restrições quanto ao tipo dos dados fornecidos pela API.

6.2.1 Iteração entre os componentes

REST prevê o uso das URIs para identificação dos recursos disponíveis na API. O separador (/) indica o relacionamento hierárquico entre esses, como no exemplo a seguir:

http://api.example.rest.org/paris/louvre/leonardo-da-vinci/mona-lisa

Uma URI deve explicitar qual o domínio dos recursos disponíveis. Segundo (MASSE, 2011) "atribuir valores significativos a cada segmento do caminho ajuda a comunicar claramente a estrutura hierárquica do modelo de recursos da API REST".

As URIs também permitem acesso a consultas parametrizadas que podem ser usadas para filtragem, paginação ou ainda busca por um termo específico dentre múltiplos de um mesmo recurso. Abaixo três

Tabela 2 – Ações para cada mapeamento URI e método HTTP

URI	Método HTTP	Ação
/book	GET	Retorna todos os livros
/book/{id}	GET	Retorna um livro específico
/book	POST	Cria um livro
/book/{id}	PUT	Atualiza um livro
/book/{id}	DELETE	Deleta um livro

exemplos de consultas através de URIs.

(1) https://api.whatsapp.com/send?phone=15551234567&text=Hello%friend

(2) GET /users?role=admin

(3) GET /users?pageSize=25&pageStartIndex=50

Em (1) uma mensagem com o texto "Hello friend"é enviada ao número "15551234567". Em (2) o retorno da consulta será uma lista filtrada com todos os usuários em que o atributo *role* é *admin*. Por fim, em (3) serão devolvidos 25 usuários a partir do de index 50.

Para interagir com um recurso é necessário, além de identificá-los com uma URI, determinar o tipo de ação que executará sobre ele. "Cada método HTTP possui uma semântica específica e bem definida no contexto do modelo de recurso da API REST."(MASSE, 2011), e explica "O objetivo do GET é recuperar uma representação do estado de um recurso. HEAD é usado para recuperar os metadados associados ao estado do recurso. PUT deve ser usado para adicionar um novo recurso ou atualizar um recurso existente. DELETE remove um recurso de seu pai. POST deve ser usado para criar um novo recurso dentro de uma coleção e executar controladores". A tabela 2 ilustra ações tomadas pela API conforme o tipo de requisição HTTP recebida.

É comum que uma API REST devolva texto como resposta a requisições, sendo que os formatos mais comuns são JSON e XML. Se a API foi implementada para suportar mais de um formato de resposta cabe à requisição informar qual delas deseja (MASSE, 2011). Além disso APIs REST utilizam links e hypermedia na representação dos dados. Com eles é possível identificar, para um recurso, as associações e ações disponíveis em um determinado estado (MASSE, 2011).

Finalmente, como benefício da arquitetura REST podemos citar também o fato dessa ser stateless, que significa que cada interação

com a API é independente e por isso deve transportar todas informações necessárias para seu processamento. Isto é, o servidor da API não armazena informações sobre o estado da conexão entre partes. Isso possibilita a replicação e balanceamento de carga, tornando a API altamente escalável.

6.3 MVC

Proposto por Trygve Reenskaug na década de 70, o padrão de projeto MVC (Model-View-Controller) tem como objetivo chave o reuso de código e a programação paralela. Em sua origem essa arquitetura era comumente usada para desenvolver GUIs (Graphical User Interfaces) (KRASNER; POPE et al., 1988), contudo se tornou popular ao ser empregada no desenvolvimento de aplicações web, móveis e desktop, assim como no caso de nossa implementação.

O conceito prevê a segmentação das responsabilidades do sistema em três partes, independentes porém conectadas - Modelo, Visão e Controlador. O **Modelo** restringe-se à manipulação dos dados (portanto interação com a camada de persistência), aplicando de lógica para cumprimento das regras de negócio. A **Visão** caracteriza-se pela representação dos dados e interação com os usuários. Enquanto o **Controlador** media as requisições recebidas pela aplicação de modo a traduzi-las em comandos para o modelo - o que, eventualmente por consequência, resulta em atualizações da visão. Ao realizar essa separação, o design da aplicação é facilitado, assim como seu desenvolvimento, uma vez que cada componente tem menos dependências com os demais (KRASNER; POPE et al., 1988).

Vale ainda a considerar a consolidação dessa estratégia dentro da engenharia de software, garantindo inúmeros Frameworks que seguem este padrão nas mais diversas linguagens de programação (LEFF; RAYFIELD, 2001), como é o caso de populares como Java, Python, Ruby e JavaScript.

Uma aplicação que respeite a esse padrão garante total independência entre front-end e back-end, o que permite que equipes distintas programem paralelamente sem precisar conhecer detalhes do outro container. Prática aplicada no processo de construção do software desse trabalho.

Uma aplicação MVC também traz como vantagens o baixo acoplamento e facilidade de incrementação - contribuindo com nosso objetivo específico de que este sistema sirva como base expansível para

implementação de futuras funcionalidades que busquem satisfazer demais necessidades administrativas de uma instituição.

6.3.1 Interação entre os componentes

A figura 1 mostra o diagrama de interação entre os componentes do MVC. O controlador recebe os comandos do usuário, valida os dados e então executa as operações para alterar o estado dos dados no modelo. O modelo responde aos comandos do controlador, atualiza seu estado e replica essa mudança para a visão. A visão então atualiza seus elementos gráficos para representar para o usuário o novo estado da aplicação.

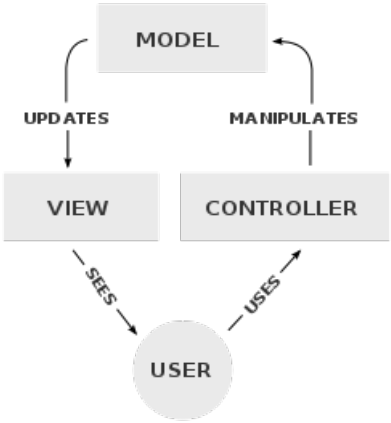

Figura 1 – Diagrama de interação dos componentes do MVC.

6.4 SINGLE PAGE APPLICATION

Single Page Application (SPA) são aplicações web que utilizam uma única página HTML como base para todas as outras páginas HTML que irão compor uma determinada aplicação. Sua implementação é feita utilizando JavaScript, CSS e HTML. "Em sua grande maioria, SPAs, são desenvolvidas no front end ao contrário de aplicações web tradicionais que dependem fortemente de interações com o servidor web

e que a cada navegação carregam novas páginas web. O comportamento e desenvolvimento de SPAs se assemelham a aplicações nativas contudo estas executam em um processo do navegador enquanto que aplicações nativas executam em seu próprio processo(FINK et al., 2014)".

SPAs funcionam sobre uma única página HTML, utilizando JavaScript, o conteúdo da página é operado para realizar as transições de uma visão para outra e lidar com navegações. "Em SPAs não é feito POST completo para o servidor, nem o recarregamento completo de uma única página web"(FINK et al., 2014). Sua interação com o servidor é majoritariamente para obter dados, estes estruturados, geralmente, no formato *json*. Outra mudança em relação aos modelos tradicionais é que SPAs armazenam seu estado no browser, dessa forma, dados de UI, por exemplo, que independem do servidor, ficam armazenado localmente, reduzindo a quantidade de requisições feitas para o servidor.

SPAs integram "o melhor dos dois mundos, aplicações web tradicionais e aplicações nativas. Ao desenvolver SPAs você obtém a portabilidade e a integração entre várias plataformas proporcionada por uma aplicação web assim como o gerenciador do estado da aplicação no lado do cliente e responsividade de uma aplicação nativa (FINK et al., 2014)".

Utilizando requisições assincrônicas e sendo responsável por grande parte da lógica da aplicação, SPAs ganham em responsividade, enquanto uma operação é realizada no servidor, o lado do cliente não fica bloqueado, dando ao usuário a experiência de interação igual a de aplicações nativas. Esses fatores "fazem das SPAs as ferramentas perfeitas para criação de aplicações web da próxima geração(FINK et al., 2014)".

Para suportar o desenvolvimento, muitas bibliotecas JavaScript foram implementas, como o React e o Redux utilizados neste trabalho. A primeira responsável pela construção de interfaces com o usuário e a segunda faz o gerenciamento do estado da aplicação no lado do cliente.

6.5 OAUTH 2.0

OAUTH 2.0 é um protocolo de autenticação padrão da indústria, definido pelo RFC 6749 (HARDT, 2012), que permite a aplicações obterem um acesso limitado às contas de usuários de serviços HTTP como Gmail, Facebook e Github.

O protocolo provê fluxos específicos de autorização para aplicações web, desktop e móveis. Esse funciona repassando o processo de autenticação do usuário para o serviço que hospeda sua conta, que, por

sua vez, capacita aplicações terceiras acessarem algumas informações do usuário, conforme sua autorização, através de um token que permite acesso aos recursos protegidos do autenticado (BIHIS, 2015).

6.5.1 Concessão de autenticação

Para realizar o processo ao qual se destina o OAUTH define quatro entidades:

- Dono do recurso:

  O dono do recurso é o usuário que autoriza a aplicação em que deseja se autenticar a acessar sua conta no serviço confiável. A aplicação tem acesso restrito a conta do usuário de acordo com as permissões cedidas por esse.

- Servidor do recurso:

  O servidor que hospeda os recursos protegidos, capaz de aceitar e responder requisições de autenticação a partir de tokens de acesso.

- Cliente:

  A aplicação que faz requisições para acesso de recursos protegidos, conforme interesse e autorização do dono do recurso. A aplicação aqui caracterizada como *Cliente* não sofre qualquer restrição por esse motivo, podendo ser executada em um servidor, localmente ou até a partir um smartphone, por exemplo.

- Servidor de autorização:

  Servidor que emite os tokens de acesso para a aplicação cliente após obter sucesso na autenticação.

Na figura 2, os quatro primeiros passos são realizados na intenção de obter a concessão de autorização e o token de acesso. O tipo de concessão depende do método utilizado pelo cliente para obter essa autorização, e dos tipos disponíveis pelo servidor de autenticação. São quatro os tipos de concessão de autorização definidos pelo OAUTH2, sendo que cada um deles atende a uma necessidade distinta.

1. **Authorization Code**: usado por aplicações na internet (web services).
2. **Implicit**: usado em aplicações para dispositivos móveis ou aplicações web, as quais executam no dispositivo do cliente.

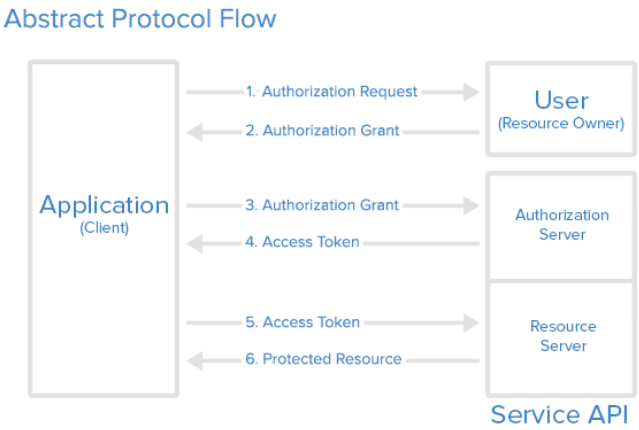

Figura 2 – Diagrama com o fluxo abstrato do protocolo OAUTH 2.0.

3. **Resource Owner Password Credentials**: usado em aplicações confiáveis, como as do próprio serviço autenticador.

4. **Client Credentials**: usado para acesso a API do cliente.

6.5.2 Outras considerações

De acordo com (MASSE, 2011) "OAUTH deve ser usado para proteger os recursos da API", pois contribui com a segurança de forma que a API REST se mantenha stateless e focada em recursos.

Outro benefício da utilização de um protocolo padrão de mercado é que possíveis erros e falhas de segurança que uma implementação de autenticação própria poderia trazer são evitados, repassando a responsabilidade para entidades consideradas competentes para prover tais serviços. Além disso, também faz parte das políticas administrativas do laboratório Bridge que os colaboradores possuam cada um uma conta do Gmail, logo permitir que a autenticação da aplicação seja feita por um recurso já existente tornaria o acesso simplificado e menos burocrático.

Portanto para garantir segurança e praticidade ao processo de autenticação do sistema desenvolvido neste trabalho foi usado do protocolo OAUTH 2.0 integrado com o serviço disponibilizado pelo Google.

6.6 QUALIDADE DE SOFTWARE

Podemos tomar qualidade como "o grau em que um conjunto de características inerentes a um produto, processo ou sistema cumpre os requisitos inicialmente estipulados para estes"(NBR, 2001). Para execução dessas medições, no que diz respeito à engenharia de software, existem duas conhecidas metodologias: Verificação e validação.

Como define (AROUCK, 2001), verificação e validação de software são métodos para avaliação de produtos e processos. Em especial, verificação trata dos aspectos técnicos, concentrando-se em qualificar se o software fora desenvolvido em conformidade com suas especificações, enquanto validação abrange as questões de negócio, provendo maneiras de estabelecer e identificar se a construção realizada atende aos propósitos qual se destina.

6.6.1 Estratégias de verificação de software

Verificação é uma técnica de qualificação dos resultados do processo de desenvolvimento de software, visando avaliar a consistência e equivalência dos artefatos produzidos durante todas as etapas anteriores de sua construção. Segundo (RTCA/EUROCAE, 1992), o propósito principal do processo de verificação de software é detectar e relatar defeitos que foram introduzidos durante o desenvolvimento deste, enquanto a correção desses defeitos fica a cargo da própria equipe de desenvolvimento.

Dentre as populares estratégias de verificação de software, a tomar os próprios sistemas desenvolvidos pelo laboratório Bridge como referência, podemos citar três práticas relevantes da literatura. São elas, teste de unidade, de integração e de sistema. Comentaremos um pouco mais sobre essas a seguir.

6.6.1.1 Teste de Unidade

Testes de unidade são designados a verificar a qualidade do funcionamento de uma parte bem isolada do programa, a costume uma única classe ou até mesmo apenas uma de suas operações. Aqui, uma unidade é a menor parte testável de um programa de computador. Espera-se que esse tipo de teste seja independente dos demais, garantindo assim isolamento do que está sendo verificado.

Costumeiramente implementados pelo próprio desenvolvedor da unidade testável, esse estágio prevê e simula os distintos contextos em que aquele fragmento do sistema virá encarar, confrontando sua resposta ao que se espera dela.

Diz-se dessa uma das mais vantajosas práticas de apoio à manutenção de software, sobretudo por favorecer a percepção ágil dos impactos causados por evoluções ou correções à dada unidade dos sistema.

6.6.1.2 Teste de Integração

Teste de integração é a fase da verificação de software em que módulos são combinados para serem testados em grupo. Ela vem depois dos testes unitários, quando o funcionamento individual de cada estrutura já foi qualificado.

O propósito do teste de integração é verificar os requisitos funcionais do sistema, avaliando também questões de infraestrutura, como performance e confiabilidade do modelo da aplicação.

6.6.1.3 Teste de Sistema

Por sua vez, testes de sistema se propõe a avaliar o correto funcionamento do produto como um todo, conforme descreve sua especificação. É um teste final, considerando a integração de todas as partes pertinentes à execução da aplicação, em simulação de um ambiente de produção.

Essa etapa tem por finalidade avaliar, em principal, iteração com a interface gráfica e integridade do sistema, submetendo-o a situações adversas e acompanhando seu comportamento, focando, portanto, nos requisitos não-funcionais da aplicação, a citar, usabilidade, acessibilidade, segurança, sobrecarga e desempenho.

6.6.2 Validação de software

A validação de software é um processo que visa elaborar evidências de que o sistema atende aos propósitos que fora designado. Diz bastante respeito a estabelecer métricas que possam explicitar o alcance daquilo que se desejava, isso é, avaliar se o que foi entregue atende às expectativas do cliente.

Em essência, serve para assegurar que, independente do que havia se planejado, a implementação satisfaz às regras de negócio e corresponde à vontade do cliente. Logo esses parâmetros são avaliados com sua participação.

Dentre diversos métodos, um dos menos burocráticos é a aplicação de testes de aceitação ao longo dos estágios finais de desenvolvimento, procedimento realizado neste projeto.

6.6.2.1 Teste de aceitação

Conforme descreve (MYERS, 2004), testes de aceitação são geralmente realizados por um grupo seleto de usuários finais, ou ainda pelo próprio contratante, num ambiente o mais próximo do qual usarão o sistema em produção. Esses simulam operações rotineiras no sistema de modo a qualificar se seu comportamento está de acordo com o solicitado e se isso compreende às atuais necessidades do cliente.

7 FERRAMENTAS

Nesta seção são descritas as principais ferramentas adotadas no desenvolvimento deste projeto.

7.1 SPRING

Spring é um framework open source que facilita a implementação de aplicações JEE (Java Platform, Enterprise Edition), ele tem como objetivo dar suporte a implementação e execução de softwares corporativos e é flexível o bastante para lidar com distintas arquiteturas de projetos (Spring, 2018).

Lançado em 2002 e utilizado em grandes projetos, o Spring é uma solução consolidada no universo de desenvolvimento de projetos Java. Propondo versatilidade, o framework é segmentado em módulos, o que permite aos programadores escolherem quais partes desejam utilizar em sua aplicação. Integrando com a plataforma é possível implementar desde simples APIs REST até grandes aplicações escaláveis que suportam acesso global através da cloud.

Para esse projeto em especifico foram utilizados os módulos Spring Boot, Spring Security e Spring Web MVC, quais serão detalhados abaixo.

7.1.1 Spring Boot

O Spring Boot é um projeto cujo objetivo é fornecer um conjunto de ferramentas que auxiliam a construção de aplicativos Spring com fácil configuração e "prontos para executar"(Spring Boot, 2018).

Sua configuração é opinativa e é por meio de anotações que se faz possível alterá-la para que atenda a requisitos específicos não satisfeitos pela configuração padrão. Neste projeto foi utilizado principalmente para configuração do ambiente de desenvolvimento e de produção, bem como a estrutura de acesso e comunicação com o banco de dados, além de configurações de segurança - como exemplo, a visibilidade dos endpoints da API disponibilizada.

7.1.2 Spring Web MVC

Spring Web MVC é um framework construído sobre a API Servlet, qual oferece a implementação das funcionalidades de um servidor web. Este é arquitetado seguindo o padrão de projeto *front controller pattern*, no qual um Servlet central coordena as requisições recebidas delegando-as para componentes configuráveis. Organização essa que permite ao framework atender diversos fluxos de trabalhos (Spring Web MVC, 2018).

A ferramenta permite tornar o mapeamento de requisições HTTP para delegadores específicos trivial, com o uso de anotações, facilitando significativamente essa tarefa indispensável no desenvolvimento de uma API REST.

Para mapear que uma URI seja direcionada para um dado controlador basta usar a anotação *@RequestMapping* que aceita como parâmetro o tipo do request HTTP (GET ou POST) conforme demonstrado na figura 3. A construção da resposta HTTP da API também fica por conta do framework que permite, com facilidade, a adição de um JSON em seu conteúdo ou a indicação de um ocasional erro.

```
@RequestMapping(value = "/api/schedule", method = RequestMethod.GET)
public ResponseEntity<Collection<ScheduleDto>> list(BaseFilter filter) {
    return this.rest.findAll(() -> this.service.load(filter)).ok();
}
```

Figura 3 – Mapeamento de uma requisição HTTP GET para uri */api/schedule* no Spring MVC.

7.1.3 Spring Security

Spring Security é um framework poderoso e altamente configurável com o foco em prover autorização e autenticação para aplicações Java. O verdadeiro poder desta ferramenta está na capacidade de sua extensão para atender a requisitos específicos de segurança (Spring Security, 2018).

Dentre diversas funcionalidades disponíveis, foram utilizadas o suporte ao OAUTH2, a integração com o Spring Web MVC e a proteção contra ataques - como os de sequestro de sessão ou CSRF (falsificação de solicitação entre sites).

A utilização do framework possibilitou atender, sem grandes dificuldades, ao requisito de autenticação através do Google por meio do OAUTH2. Além disso esse possibilita que o acesso aos dados do usuário, obtidos pela autenticação, sejam simplificados por meio de anotações, permitindo sua utilização em qualquer local da aplicação.

7.2 JAVA TIME

Muito importante para a estratégia adotada como solução das questões de negócio associadas ao software desenvolvido foram as estruturas disponibilizadas no package Time, incorporado a partir do Java Standard Edition Development Kit (JDK) 8. O pacote oferece uma sólida representação de inúmeros aspectos do tempo pertinentes para a composição desejada, citaremos a seguir os mais importantes:

- **TemporalUnit**

  Uma das mais fundamentais abstrações sobre a representação do tempo é a capacidade de medi-lo. TemporalUnit representa uma unidade de medida em si, tal qual temos segundo, semana ou ainda século.

- **TemporalAmount**

  Em adição, adquire-se o conceito de uma medida especifica em base dessas unidades - como uma idade (24 anos, 3 meses e 8 dias) ou uma duração (8 minutos e 1672243 nanosegundos).

- **TemporalField**

  Já a representação de um momento, para os humanos, é expressa usando campos que fracionam a linha do tempo com base em algo que nos tem significado, como os anos, dias do mês, horas do dia, etc. Essa interface modela a representação destes campos.

- **Temporal**

  Por fim, aqui temos uma estrutura capaz de determinar um específico ponto no tempo, como uma certa data, uma precisa hora, sobre as quais é possível obter informações a respeito dos campos que as compõem, sua relação temporal com outros objetos de mesmo tipo, ou ainda manipulação através de operações temporais ou edições diretas.

- **Queries e Adjusters**

> Também vale mencionar a respeito da arquitetura de consulta e manipulação proposta pelo pacote que permite bastante flexibilidade, praticidade e esclarecimento no tratamento desses objetos temporais.

Para além das interfaces apresentadas, o java.time conta com implementações muito bem fundadas sobre as principais utilizações humanas do tempo, como datas, horas, instantes, períodos, fuso-horários e cronologias. Material de grande valor para o desenvolvimento de ferramentas que, tal qual o software aqui proposto, tem seu negócio tão relacionado com esse conceito que é inerente a toda vida em sociedade, o tempo.

7.3 REACT

React é uma biblioteca declarativa, eficiente e flexível, implementada em Javascript para construção de interfaces de usuário (UI). Criada pelo Facebook em 2013, tem como objetivo resolver os desafios de desenvolver UIs complexas com dados que mudam o tempo todo (GACKENHEIMER, 2015).

A ferramenta propõe a modularização da interface gráfica da aplicação, segmentando sua construção através de múltiplos componentes, os quais integrados compõem uma UI complexa. Cada componente é responsável por representar seu próprio conjunto de dados, chamado estado, que ao sofrer qualquer alteração renderiza o componente novamente, mantendo sempre a exibição do estado mais atual dos dados.

Ao sofrerem mudança, os componentes, representados por um DOM (Document Object Model), precisam ser alterados; E a forma mais simples de realizar isso seria removendo o nodo referente ao componente e depois adicionando-o novamente em seu novo estado. Porém, para otimizar esse processo, a biblioteca React trabalha com DOM virtual que a permite calcular de forma eficiente a mudança ocorrida, aplicando, então, apenas a alteração necessária no DOM efetivo (Pete Hunt, 2013).

A biblioteca possibilita ainda o desenvolvimento de aplicações ditas como Single Page Application, assim como o de aplicativos para dispositivos móveis através do React Native. Além de possuir integração com as principais bibliotecas destinadas ao desenvolvimento de aplicações que executam no lado do cliente, como é o caso do Redux, também utilizado em nossa implementação.

7.3.1 Bridge-React

Com o intuito de acelerar o processo de desenvolvimento de aplicações da instituição e atender aos requisitos de design, foi criado pelo Laboratório Bridge uma biblioteca de componentes react, denominada bridge-react. Esta biblioteca contém os componentes básicos de UI, como botões, listas, containers, etc e foi construída respeitando o Design System proposto pela equipe de design do laboratório Bridge. O uso da biblioteca foi fundamental para o desenvolvimento deste trabalho, possibilitou que o mesmo fosse desenvolvido em tempo hábil e respeitando as características de design do laboratório. A flexibilidade proporcionada pela biblioteca possibilitou customizá-los para atender aos requisitos específicos deste projeto diminuindo consideravelmente a necessidade de criação de novos componentes.

7.4 REDUX

Redux é uma biblioteca para gerenciamento do estado de aplicações Javascript. Com requisitos cada vez mais complicados, as single-page applications em JavaScript, requerem, de forma jamais vista antes, que desenvolvedores escrevam código para gerenciamento do estado da aplicação (Dan Abramov, 2018).

O estado da aplicação pode conter respostas do servidor, dados em cache, dados criados localmente e que ainda não foram persistidos no servidor. Além disso o estado de UI tem se tornado cada vez mais complexo, sendo necessário lidar com gerenciamento de rotas ativas, abas selecionadas, spinners, controle de paginação, etc. Tratar este estado que está em constante mudança é uma tarefa difícil (Dan Abramov, 2018).

Buscando resolver esses problemas, Dan Abramov, criou o Redux. Redux utiliza uma fonte de única de verdade, a *store*, que é o estado de toda a aplicação. A *store* pode apenas ser lida e para modificá-la é necessário descrever essas mudanças em objetos planos chamados *Actions*.

Usando esta estratégia a biblioteca é capaz de lidar com diferentes domínios da aplicação e múltiplos estados de componentes gráficos tudo isso mantendo a imutabilidade dos dados e resolvendo comportamentos assíncronos. A biblioteca facilita também a adição de nova funcionalidades na aplicação assim como a realização de testes.

Redux pode ser utilizado com diferentes bibliotecas de interface

de usuário em JavaScript, inclusive o React. A biblioteca responsável por isso é React-Redux, ela permite conectar a *store* do Redux no estado de um componente React. Quando a *store* é alterada o estado do componente também é, atualizando-o imediatamente para representar graficamente o estado mais atual da aplicação.

8 LEVANTAMENTO DE REQUISITOS

A etapa de levantamento de requisitos é um importante componente do processo de análise na engenharia de um software. Constatada a viabilidade de uma implementação, entendido o problema a se solucionar, é através de contato com o cliente que se acorda especificações a respeito das funcionalidades e infraestruturas que o sistema final deve prover. Essas definições costumam por compor uma documentação do software que tem por objetivo descrever formalmente seu funcionamento esperado e com isso estabelecer com clareza escopos de implementação negociados, bem como permitir a verificação de seu correto desenvolvimento.

Para nosso projeto, a gerência administrativa do laboratório Bridge foi tomada como cliente, tanto por ser grande interessada na utilização de um sistema para os fins que este se propõe, quanto pelo amadurecimento adquirido ao longo dos anos através da experiência dos usuários com as ferramentas que esse substituirá. E, portanto, possuindo aptidão para realizar os devidos esclarecimentos e negociações provenientes dessa implementação.

No percurso de nosso desenvolvimento o contato com o cliente foi intenso e frequente, prática muito valorizada pelo manifesto ágil (Kent Beck et al., 2001), qual indica que "Pessoas relacionadas ao negócios e os desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto". Essa dinâmica possibilitou que os rumos tomados nas implementações seguissem em progresso contínuo, sem que drásticas mudanças não esperadas gerassem retrabalho.

Como resultado desse processo, disponibilizamos os principais requisitos satisfeitos na versão em produção:

8.1 GERAL

- Cada colaborador deverá acessar o sistema através de seu e-mail institucional (de domínio bridge.ufsc.br).
- Apenas o próprio colaborador deve ser capaz de alterar suas informações.
- O calendário de expediente da organização deve ser acessível para qualquer usuário.

- Disponibilizar hints para campos / ações relevantes.

8.2 HORÁRIO SEMANAL

- Os colaboradores devem ser capazes de declarar seu horário de trabalho semanal
- Deve ser possível tratar mudança na carga horária de um colaborador informando início da vigência novo horário.

8.3 REGISTRO DE ATIVIDADE

- Cada colaborador deve ser capaz de diariamente registrar seus períodos de atividade de trabalho realizados para aquele dia.
- Deve ficar apresentado o somatório de horas realizada por dia, indicando caso falte ou exceda tempo em relação ao planejado.
- No preenchimento do horário, assumir o formato hh:mm, sem que o usuário precise colocar os 2 pontos.

8.4 CONTROLE DE AUSÊNCIAS

- Deve ser possível ao colaborador registrar uma ausência e, a depender de seu motivo, as horas de trabalho daquele dia não precisarão ser compensadas.

8.5 GESTÃO DE FÉRIAS

- Deve ficar apresentado ao colaborador quanto de férias esse tem direito
- O colaborador deve poder definir quando gozará das férias que possui

9 TIVEMOS

Dentro do universo da informática a, Graphics User Interface (GUI), diz respeito a camada gráfica que possibilita a interação entre um sistema e seus usuários, isso é, é através dessa que se comunicam os humanos e os dispositivos digitais. Desta maneira sua composição é de enorme significância no desenvolvimento de aplicações em geral, mas especialmente naquelas cuja utilização se destina a uma grande variedade de públicos. Por esse motivo pesquisas e investimentos nessa áreas tem sido recorrentes, garantindo em grandes projetos de software equipes dedicadas a essas decisões.

Para a construção de uma interface gráfica adequada a um software é preciso conhecer sobretudo seus usuários alvo, considerar como esses utilizarão o sistema, quais objetivos pretendem atingir com seu uso e, portanto, como favorecer a utilização das principais ações proporcionadas pela aplicação. Indo mais adiante, a conceitualização sobre o uso de uma aplicação com o objetivo de elaborar propostas para sua interface tem por responsabilidade considerar ainda as condições físicas de uso daquela aplicação, o nível de instrução daqueles que a utilizarão, seu conhecimento sobre o negócio, possíveis deficiências que possuam e quaisquer outras situações que venham a influenciar a experiência de interação entre o software e seus usuários.

9.1 BRIDGE DESIGN SYSTEM

Logo, foi experienciando essas importâncias no desenvolvimento de seus projetos que o laboratório Bridge passou a dedicar parte da sua equipe para estudo e elaboração de um conceito e ferramenta open source para a construção de aplicações front-end que, com o objetivo de reunir o estado da arte no que diz respeito a pesquisa e tecnologia dessa tarefa. Tivemos, primordialmente, um time de analistas e designers determinando, em consideração a todos os pontos significativos citados, qual deveria ser o resultado visual e comportamental dos componentes, seguidos pela implementação dessas construções por uma equipe de programadores.

Como resultado tivemos fundamentação do, então chamado, Bridge Design System, qual se compõe por um conceito de interface, sua coleção gráfica de componentes de design, com discriminação dos seus estágios visuais e uma descrição formal de seus funcionamentos. Além

ainda de uma biblioteca com a implementação desses em React, nomeada Bridge React, auxiliado por um site para consulta sobre o funcionamento e uso das estruturas.

Toda essa infraestrutura proporcionou base para a modelagem e implementação da interface gráfica empregada para o software produzido por este trabalho, qual, por sua vez, serviu também como um dos primeiros exemplares da utilização da plataforma, iniciando a difusão dessa estratégia de desenvolvimento dentro da instituição.

9.2 IDENTIDADE VISUAL

Em complemento, no último ano, o laboratório Bridge tomou a iniciativa de construir uma identidade visual própria com o intuito de melhorar sua comunicação interna e externa, transmitindo assim de maneira mais efetiva seus valores institucionais e postura organizacional. Este amadurecimento influenciou desde o modelo de comunicados internos, seus canais, sua linguagem, assim como as redes sociais e site da instituição.

De mesma maneira, por ser mais uma frente comunicativa da organização, se intuiu que a aplicação aqui proposta seguisse os conceitos da identidade visual desenvolvida. Sendo assim foi realizado, em colaboração com a equipe de design, um estudo dedicado à construção da interface deste sistema, qual fez parte dos esforços relacionados ao projeto de conclusão de curso da graduanda da área e colaboradora do laboratório Gabriela de Mattos.

> "Quando bem estruturada, com canais e formatos definidos, a comunicação interna reforça valores institucionais, aumenta o nível de confiança dos servidores e faz com que eles atuem como embaixadores da instituição dentro e fora dela, pois aumenta sua sensação de pertencimento."(MATTOS, 2017).

10PROCESSO DE TRABALHO

Realizado o levantamento de requisitos e esclarecidos os objetivos da aplicação em suas diversas dimensões de uso, se estudou propostas para os fluxos de interação com as funcionalidades desejadas, seguindo os preceitos já estabelecidos pelo sistema de design do laboratório e guiado pelas características de sua identidade visual. Em sequência, ao passo que uma das estratégias levantadas se firmou, foram prototipadas as primeiras telas e seu fluxo figurado com o apoio de ferramentas dedicadas.

A partir desse material ocorreu incrementalmente o alinhamento de todas as partes envolvidas quanto as expectativas sobre o funcionamento da ferramenta final. Cliente, analistas, designers, programadores, testers e usuários participantes acordaram prazos, escopos e investimentos assumidos por cada uma das partes, designando como ocorreria o processo de colaboração para construção do software idealizado.

Fora então dado início a constituição da arquitetura de dados da aplicação, sucedida, por fim, pela efetiva codificação dos produtos de software proventos deste trabalho acadêmico. Para tal, a construção do serviço disponibilizado fora paralelizado, separando os esforços do desenvolvimento de suas disjuntas camadas de front-end e back-end. Isto é, as implementações referentes à API REST e à aplicação web prosseguiram de maneira simultânea mantendo sincronia e integração durante esse processo.

Dessa maneira, iterativamente para cada grupo de requisitos associados a uma das funcionalidades do sistema, fora construído no back-end as estruturas pertinentes a aquele contexto de negócio, acompanhados de seus devidos testes, ao passo que eram concretizadas no front-end as telas provedoras dos mesmos serviços, concluindo cada ciclo pela integração das interfaces com os correspondentes endpoints da API.

A disjunção dessas implementações foi aspecto verdadeiramente significante para o processo de desenvolvimento do trabalho, permitindo completa independência entre as decisões estabelecidas no interior de cada uma das camadas, além de possibilitar a paralelização de seus desenvolvimentos. Essa característica proveniente do fundamental desacoplamento entre as partes se constitui, por fim, por essas comporem distintos softwares, cuja bem definida interação possibilita que integrados funcionem, a visão externa, como uma unidade.

A cada ciclo, portanto, um conjunto de requisitos era elencado e o planejamento para suas satisfações eram revistos, em diversas vezes implicando em minuciosos ajustes nas interfaces e nos fluxos de interação com as funcionalidades. Sendo assim, a comunicação com a designer responsável foi frequente durante toda a extensão do trabalho, sua participação nas discussões e decisões sobre a interface da aplicação foi de imprescindível relevância para o rumo que o desenvolvimento, sobretudo do front-end, tomou.

Em adição, o fácil acesso ao cliente também nos permitiu definir com agilidade quaisquer iminentes mudanças na modelagem, necessárias devido a dificuldades de implementação encontradas com o projeto em andamento. Esse fluxo de trabalho possibilitou que, mediante a essas situações, pudéssemos tratar corretamente da reavaliação das decisões envolvendo todas as esferas participantes dessa construção, conforme previsto para um ciclo de engenharia de software. Neste ponto também, as estratégias adotadas para os testes da aplicação foram fundamentais para garantir agilidade na identificação de instabilidades proveniente das mudanças realizadas.

11IMPLEMENTAÇÃO

11.1 ARQUITETURA DE DADOS

Aplicações computacionais operam sobre dados e, em sua imensa maioria, armazenam esses por meio de tecnologias que garantem a persistência dessas informações através de interrupções energéticas. O modo como esses dados são organizados implica fundamentalmente na performance de seu consumo, manejo e escrita, portanto más escolhas na construção da arquitetura de uma aplicação acarretam em seu baixo desempenho, restrita escalabilidade ou até na impossibilidade de execução de certas funcionalidades.

11.1.1 Modelo Conceitual

Com isso em vista, iniciamos a concepção de nossa organização de dados dando atenção às principais entidades envolvidas nos serviços que desejamos disponibilizar. Dentre elas, essencial mencionar o Colaborador, objeto central do proposito desde software, relacionando-se tanto com as funcionalidades propostas neste trabalho quanto com as pensadas para futuras etapas de expansão do sistema. Ao Colaborador associamos uma diversidade de atributos, desde informações pessoais à características organizacionais, de modo a estruturar, primeiramente, um repositório centralizado e completo acerca das pessoas associadas ao laboratório (ou qualquer instituição que o sistema possa servir).

Em sequência elencamos Horário, Registro de atividade e Afastamento como entidades presentes em nosso modelo. A primeira representa os períodos em que o colaborador se comprometeu com a execução das suas tarefas, a segunda identifica os trechos nos quais o usuário efetivamente esteve a serviço da organização e a terceira armazena intervalos de inatividade do participante. Por último, figurando feriados e outras datas relevantes à organização, surge o Evento.

Realizada algumas iterações de revisão e complementação do modelo, chegamos à seguinte modelagem conceitual:

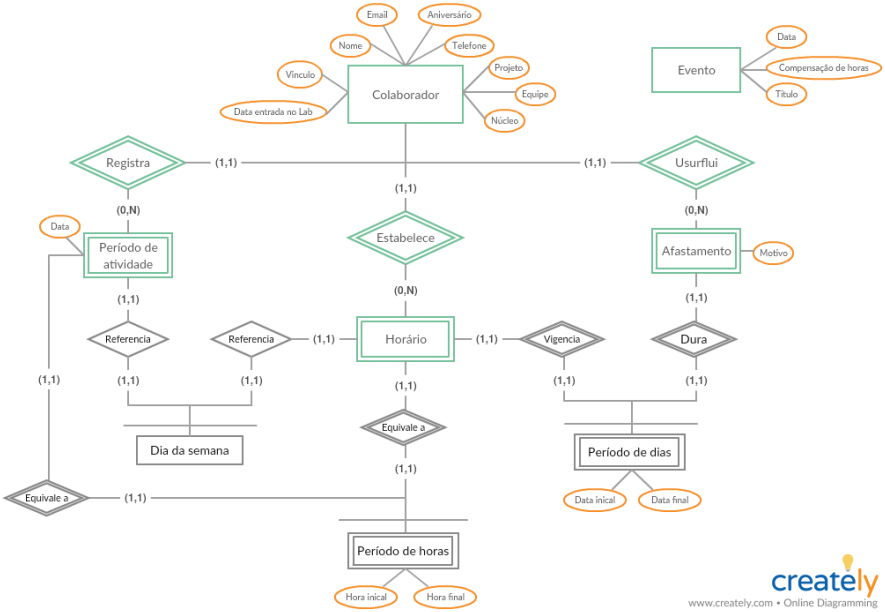

Figura 4 – Modelo conceitual da arquitetura de dados

11.1.2 Modelo Lógico

Satisfeitos com o alcançado na etapa anterior, partimos para a construção do projeto de modelo lógico de nosso banco de dados. Nessa fase ocorre a definição de tipagem e tamanho de atributos, identificação das estruturas de relacionamento entre os objetos, e basicamente o mapeamento de cada entidade conceitual para uma ou mais tabelas físicas.

Para o colaborador, por exemplo, e seus atributos telefone e e-mail, quais, por decisão de projeto, poderão ser inúmeros, se faz parte desse estágio a criação de uma tabela dedicada ao relacionamento entidade-atributos de modo que a quantidade de cada um desses associado a um mesmo colaborador não esteja pré-definida pelo número de colunas disponíveis.

De maneira semelhante, a relação entre colaborador e seu(s) projeto(s), equipe(s) e núcleo(s) também passa a ser armazenada em uma tabela dedicada que ganha a função de descrever genéricos agrupa-

mentos entre os usuários do sistema. Isso tem por objetivo favorecer a filtragem de colaboradores nas funções de consulta de suas informações.

11.2 ARQUITETURA DO BACK-END

Para a organização do funcionamento lógico por trás da API REST disponibilizada, fora adotado um padrão de arquitetura que visa segmentar as responsabilidades da aplicação em camadas bem definidas, auxiliando assim seu entendimento e portanto sua evolução; proporcionando independência entre etapas decorrentes de uma ação, logo simplificando sua manutenção e favorecendo a reutilização de código; e ainda reforçando o esclarecimento das tarefas de cada objeto. Dentre os padrões incorporados podemos citar:

- **Controller**

  Responsável por interpretar as requisições recebidas pelo servidor, essas classes satisfazem as designações de controle definidas no MVC, servindo como camada de gerenciamento para acesso da aplicação. Apoiadas pelas facilidades proporcionadas pelo uso do Spring, classes desse tipo são anotadas para executar o roteamento das requisições recebidas para os correspondentes serviços.

  Para tal, é checada nessa etapa a autorização de acesso do requisitante àquela função, bem como a integridade dos parâmetros passados, prosseguindo apenas em casos de sucesso, devolvendo uma resposta de erro quando encontrado algum problema.

- **Validator**

  A camada de validação tem por objetivo garantir consistência de objetos do domínio, para isso podem ser conferidos obrigatoriedade de atributos, validade de seus valores, existência de estruturas referenciadas, coerência entre os dados, entre outros. Em caso de invalidade são elencados todos os pontos de inconsistência como retorno da validação.

- **Service**

  Os Services servem de fachada para o modelo da aplicação, reunindo todas as funções providas para seu determinado contexto. Lhes cabe proporcionar os diversos serviços de consulta e manipulação das estruturas pertinentes ao domínio de negócio do software, gerenciando os demais componentes do modelo para execução das tarefas que se propõe, de modo a garantir a consistência

do banco de dados durante suas ações por meio do tratamento das transações que abre com esse.

- **Command**

  Idealizando classes que sejam especialistas na execução de uma específica ação independente, o padrão de projeto Command objetiva modularizar as distintas investidas que possam ser realizar sobre os objetos do domínio, favorecendo a reutilização de código, contenção de falhas e definição de responsabilidades. Essa prática contribui ainda para uma excelente organização do código, contendo os métodos de uma macro ação em uma classe por essa especializada.

- **Query**

  Construções do tipo Query são designadas a prover organização para o acesso ao banco de dados realizado pela aplicação. Essas mapeiam ao nível da linguagem as diferentes consultas que uma determinada estrutura pode sofrer, disponibilizando dentro do modelo claras chamadas de métodos para deste consumir de maneira simplificada as informações persistidas.

  Funcionando quase como uma especialização do padrão Command, também fica a cargo de uma Query transcrever as solicitações para a adequada linguagem de consulta de dados utilizada pelo banco de dados associado, gerenciando então as filtragem de colunas e combinações de tabelas para a correta realização de suas requisições, de acordo com os parâmetros encaminhados a essa.

- **Entity**

  Uma classe identificada como Entity serve como direta representação de uma das tabelas do banco de dados. Essa composição tem como objetivo estruturar as informações persistidas enquanto operadas na memória, de forma que um objeto dessa classe equivale a uma especifica linha dentro da determinada tabela do banco de dados.

  Tal estratégia, em conformidade com a utilização da JPA (Java Persistence API), provê facilidade para execução das tarefas associadas ao consumo e persistência de informações. A adoção dessa técnica torna trivial as principais operações sobre estruturas do banco de dados.

- **DTO**

  Como o próprio nome já diz, Data Transfer Object (DTO) contitui uma estrutura designada à transferência de um conjunto de dados através da aplicação. Esse padrão de projeto traz como preceito a primitividade da classe, responsabilizando-se apenas por prover getters e setters de seus atributos, sendo, portanto, incapazes de executar qualquer tipo de processamento sobre as informações que armazenam. Objetos desse tipo são vastamente utilizados para conglomerar múltiplas informações resultantes de um processamento.

11.3 AUTENTICAÇÃO

Como requisito dessa aplicação tem-se o de realizar autenticação utilizando o Google, para satisfaze-lo foi utilizado o protocolo OAUTH2. O Google implementa um servidor de autorização para esse protocolo e o disponibiliza de forma gratuita por meio de uma API.

Para utilizar a API é necessário cadastrar as credenciais do cliente no site do Google. Ao se cadastrar, é disponibilizado o id do cliente e uma senha de acesso. Feito isso já é possível utilizar o serviço do Google para realizar autenticação em uma aplicação.

O Spring, integrado com o módulo Spring Security, oferece a implementação do protocolo OAUTH2 simplificando o processo de desenvolvimento da autenticação no servidor. O primeiro passo é adicionar no arquivo .properties o id do cliente e o segredo obtidos no cadastro da credencial. O Spring possibilita a criação de diferentes perfis, cada um representado por seu .properties. Para este projeto dois perfis foram criados, desenvolvimento e produção. Cada um destes utilizam uma credencial distinta para autenticação, contribuindo para a autonomia no desenvolvimento já que este independe da credencial de produção.

Para habilitar o OAUTH2 na aplicação o Spring disponibiliza uma interface *WebSecurityConfigurerAdapter*, nela além de configurar o OAUTH2, também foram feitas as configurações de quais endpoints são públicos ou privados, qual comportamento terá o servidor ao receber requisições ainda não autenticadas, url de redirecionamento após sucesso na autenticação, assim como proteção contra ataques, como o CRSF. Com essas configurações prontas o servidor já consegue se comunicar com o Google e autorizar os acessos na API REST.

Entretanto, o serviço de autenticação do Google apenas valida se o login e senha de um usuário cadastrado na plataforma são válidos.

Porém deseja-se restringir o uso desta aplicação em específico apenas para colaboradores do Laboratório Bridge. Para isso foi utilizado o domínio de e-mail pertencente ao laboratório(bridge.ufsc.br). Como a API do Google não permite restringir quais domínios podem receber autorização, isso foi feito no servidor. O Spring disponibiliza a interface *AuthenticationProvider* que permite customizar o processo de validação da autenticação no servidor. Nela foi então implementada a verificação se o usuário autenticado possui o e-mail do laboratório.

Visando simplificar e facilitar o acesso a plataforma, foi adotado para um primeiro contato, a criação automática de um usuário simplificado, quando este realiza o primeiro login na aplicação. Para isso alguns dados básicos foram necessários, como nome, e-mail e imagem do perfil. Estes dados são fornecidos pelo Google após o sucesso na autenticação. O Spring popula, de forma transparente para o desenvolvedor, a classe *DefaultOAuth2User* com os dados fornecidos pelo Google, tornando trivial a obtenção dos mesmos na aplicação.

Por fim, ao receber uma requisição na API REST, deseja-se validar se o usuário que a realizou possui autorização para o recurso requerido. O Spring facilita essa tarefa, injetando o objeto *OAuth2AuthenticationToken*, em qualquer método da aplicação que o tenha como parâmetro. Esse objeto possui todas as informações do atual usuário logado, inclusive os necessários para a verificação de permissão.

11.4 BANCO DE HORAS

Para o cálculo do banco de horas de um colaborador é considerado, a partir de sua escala de trabalho semanal, a previsão diária de horas dedicadas à organização. Em complemento o registro dos períodos em que esse exerceu suas atividades; E por fim, ausências não-compensativas, como feriados, dias de trabalho abonados ou ainda a retirada de férias.

11.4.1 Horário semanal

A começar por registrar seu planejamento semanal de horário de trabalho, o usuário pode declarar, para cada dia da semana, quais trechos do dia estará disponível para execução de suas responsabilidades com a organização. Ele pode ainda determinar a vigência daquela escala, mudando seu horário e gerenciando histórico de registros com

facilidade.

Tal declaração tem dois principais objetivos. Primordialmente disponibilizar essa informação para consulta de qualquer interessado, proporcionando um ambiente centralizado e confiável sobre o horário de trabalho daquele colega. Em complemento, o volume total de atividade diário registrado é considerado para o cálculo do banco de horas do colaborador de modo a debitar tal valor planejado no iniciar de cada dia.

11.4.2 Registro de atividade

Também é possível, para cada dia de expediente, registrar os trechos em que executou seu trabalho, acompanhando em tempo real o comparativo entre a expectativa planejada e o efetivo realizado, tanto para o dia corrente quanto para a semana atual, ou ainda para períodos mensais. Por fim, em adição, a consulta e manipulação do histórico de registros está disponível de mesmo modo.

A inscrição desses intervalos contribui positivamente, isso é, como saldo, para o cálculo do banco de horas do colaborador. Apenas os períodos datados até o dia corrente são considerados para a obtenção do valor final calculado.

11.4.3 Ausências

Fica disponibilizado ainda a oportunidade de descrever um período no qual o usuário esteve afastado das suas responsabilidades, seja por adoecimento, acordo, ou mesmo por férias. Além do indicativo visual e do registro histórico armazenado, as datas abrangidas por uma ausência são desconsideradas para a subtração de horas proveniente do horário semanal, no que diz respeito ao cálculo.

11.4.4 Eventos e Feriados

Semelhante às ausências, existe estrutura dedicada para a descrição do calendário de expediente da organização. Neste, eventos institucionais e feriados podem ser declarados, de modo a aparecerem como destaque na listagem de atividade dos usuários, bem como ter seu período abatido da expectativa de trabalho de todos os colaboradores.

11.4.5 Cálculo

O resultado do banco de horas é adquirido, portanto, operando os períodos declarados, afim de obter o intervalo equivalente à correta expectativa de trabalho de um colaborador, debitando-o este tempo. Em seguida se é somado contra este volume, o total de horas trabalhadas pelo mesmo.

Invariavelmente há inúmeras estratégias pertinentes para a solução dessa questão, outras inclusive foram cogitadas para a implementação deste trabalho, entretanto, por fim, a abordagem escolhida foi a de modelar objetos que representassem intervalos temporais capazes de serem operados em mais alto nível e, sobretudo, expandir as possibilidades de processamento dessas informações para além do cálculo aqui exposto.

11.5 BRIDGE TEMPORAL INTERVAL

Fora então implementado um pacote de classes e interfaces destinados modelar uma estrutura computacional capaz de representar trechos de tempo de modo a permitir tratá-los como conjuntos matemáticos e aplicar, portanto, verificações como a pertinência de um elemento ou sobreposição com outros conjuntos, bem como operações de união, intersecção e diferença entre tais objetos.

Em cima da concepção de tempo pertencente ao JDK SE 8, decidimos por expandir os conceitos abrangidos no pacote para incorporar a ideia de intervalos temporais, não disponibilizada de forma nativa pela linguagem, até então. As primitivas já modeladas foram excelente ponto de partida para a conceitualização desejada, o que, além de contribuir para o esclarecimento do problema, possibilitou uma construção simples, concisa, orientada, flexível e ainda poderosa, capaz de suportar a manipulação tanto das ideias presentes em nosso problema quanto para distintas finalidades.

Comentaremos as principais estruturas desenvolvidas a seguir:

- **TemporalInterval**

  Um TemporalInterval caracteriza um período no tempo, sendo delimitado pelo momento de seu início e o de sua conclusão, representando em essência o conjunto dos momentos abrangidos por essa passagem do tempo.

  Essa abstração, sendo também um Temporal, estende suas estra-

tégias de consulta e edição do objeto mantendo os preceitos de imutabilidade propostos pelo pacote nativo.

Em adição assina métodos que estruturam as principais operações realizadas sobre conjuntos matemáticos (propósito principal dessa interface). Dentre eles:

- `boolean contains(Temporal temporal);`
- `boolean intersects(Temporal temporal);`
- `TemporalInterval union(Temporal temporal);`
- `TemporalInterval intersection(Temporal temporal);`
- `TemporalInterval difference(Temporal temporal);`

O conceito aqui empregado torna-se princípio para formalização de demais ideias subsequentes.

- **Continuous e UncontinuousTemporalInterval**

  Tratando-se de trechos de tempo podemos considerar intervalos ininterruptos, fracionados ou até periódicos. Os dois primeiros casos foram considerados, resultando em interfaces que manejam distintas estratégias para a execução das operações de conjuntos inerentes de um TemporalInterval, qual extendem.

  Implementando essas interfaces temos um par de classes incluso no mesmo pacote. **ContinuousTemporalRange** segue as normas de um intervalo matemático, representando todos os elementos entre seus extremos, inclusiva ou exclusivamente. Ao passo que **UncontinuousTemporalTree** armazena múltiplos intervalos contínuos disjuntos.

- **ChronoAmount**

  Expandindo o conceito nativo simples de um TemporalAmount, esta interface constitui uma proposta mais maleável e abrangente para a representação de volumes de tempo.

  Possibilitando vasta manipulação, objetos desse tipo permitem medir uma duração em função de qualquer combinação de TemporalUnits, disponibilizando ainda maneira de normalizar essa contagem, de obter sua equivalência em fator de outras unidades de medida ou ainda operá-la matematicamente com outros TemporalAmounts;

- **ChronoInterval**

  Por sua vez, mesclando os conceitos de um conjunto de momentos e da passagem de tempo, inerentes de um intervalo temporal, ChronoInterval estende TemporalInterval e ChronoAmount e tem por propósito representar trechos no tempo no que diz respeito a seu volume e sua localidade temporal.

  As especializações Continuous e UncontinuousChronoInterval aparecem de mesma maneira e é a partir de suas implementações que dispomos de uma sólida construção para manejo de problemas envolvendo cronológicas passagens do tempo.

- **TemporalDirection**

  Para identificação das extremidades de um TemporalInterval e, em extensão, a relação temporal entre Temporals, se complementa as convenções matemáticas do pacote nativo, qual indica pontos no tempo através de representações numerais, constituindo a relação de passado e futuro entre os objetos através da reta dos números inteiros.

  A enumeração TemporalDirection consolida de maneira mais determinada as instancias que representam essas relações temporais, a incluir também o presente, trazendo-os como Temporals para inúmeros fins, bem como possibilitando operações numerais pertinentes sobre valores representados nesse domínio.

11.5.1 Considerações sobre a implementação

Por se tratar de uma proposta de extensão da arquitetura disposta pelo pacote Java.Time, diversos dos princípios do pacote foram considerados para construção dessa biblioteca. Além da representatividade numeral de objetos temporais, da nomenclatura dos métodos e de dinâmicas da interação entre os objetos, as principais estratégias algorítmicas identificadas foram mantidas, buscando dispor uma solução coerente com a proposta da linguagem e de simples uso e entendimento para aqueles que já conhecerem do modelo base.

11.5.1.1 Imutabilidade

Um aspecto predominante em todas as classes temporais disponíveis pela linguagem é o de não edição de seus atributos. Em vez

disso, métodos que possibilitam a manipulação dos objetos tem por resultado a criação de uma nova instância que difere da atual pela a dada alteração encaminhada.

Essa abordagem permite uma melhora de performance para utilização de threads, já que o estado dos objetos nunca muda e, portanto, múltiplas threads podem acessá-los sem restrição. Também contribui significantemente para uso de chaves em estruturas de dados, visto que a alteração do objeto que foi usado como chave pode resultar em mudança de toda a estrutura de dados, o que não acontece aqui.

Esse padrão foi rigorosamente mantido na implementação das classes presentes em nossa biblioteca, tal qual é fortemente recomendado na documentação das estruturas do pacote Java.Time.

11.5.1.2 Delegação de responsabilidade

Uma das estratégias é a de que, quando da interação entre objetos Temporais, certas chamadas de método de uma classe possuem um método oposto equivalente no objeto que com essa está interagindo. Dessa maneira, as classes são capazes de resolver as questões dentro de um universo de iteradores conhecidos. E seja o caso de desconhecê-los repassam a responsabilidade de execução para a outra classe.

Como exemplo, a intersecção entre um intervalo contínuo e um incontínuo não pode ser resolvida pelo primeiro deles, qual delega a obtenção da resposta ao par, tornando as seguintes chamadas equivalentes:

```
continuous.intersection(uncontinuous);
uncontinuous.intersection(continuous);
```

De mesma modo, um intervalo incontínuo que seja encarregado de responder sua intersecção com TemporalInterval qual não conhece especializações vai delegar a esse tal responsabilidade de resposta.

11.5.2 Importância para o trabalho

Tratar dos múltiplos intervalos de tempo presentes no domínio do nosso negócio através das estruturas citadas possibilitou manejar as conjunturas necessárias com bastante simplicidade. A tomar o cálculo do banco de horas como exemplo, foram-se utilizadas das operações

de conjuntos para obtenção do intervalo temporal equivalente à real expectativa de trabalho do colaborador, através da diferença entre seu planejamento de trabalho semanal e os atestados, feriados e férias do mesmo período. Em seguida, o suporte à soma do volume de tempo de tais intervalos também tornou simples o confronto entre o débito de tempo calculado e o saldo proveniente dos registros de atividade do colaborador, resultando por fim na obtenção do valor desejado.

11.6 API REST

Como resultado do desenvolvimento do servidor, temos uma API REST, abaixo são listados os endpoinds disponibilizados pela mesma e suas respectivas funções.

- /*

  GET - retorna o index HTML contendo a aplicação do frontend

- /api/absence

  GET - retorna as ausências do colaborador
  POST - adiciona uma ausência para um colaborador
  PUT - edita uma ausência de um colaborador

- /api/activity

  GET - retorna as atividades diárias
  POST - adiciona uma ou mais atividades diárias
  PUT - edita uma ou mais atividades diárias

- /api/balance

  GET - retorna o banco de horas de um colaborador

- /api/schedule

GET - retorna escalas de horário

POST - adiciona uma escala de horário

PUT - edita uma escala de horário

- /api/user/current

  GET - retorna o atual usuário logado na aplicação

11.7 APLICAÇÃO WEB

Implementado como single page application, o frontend fica encarregado de ditar o fluxo da aplicação assim como coordenar as requisições de dados para a API REST. Para sua construção duas bibliotecas foram fundamentais React e Redux. React utilizado na implementação das interfaces de usuário e Redux coordenando todo o estado da aplicação no lado do cliente.

A arquitetura recomendada para o desenvolvimento de aplicações que utilizam estas bibliotecas é a segmentação entre View e Container. View é responsável pela construção de componentes gráficos e as operações necessárias para o funcionamento do mesmo. Toda a lógica do modelo, busca de dados no servidor e armazenado destes ficam sob a responsabilidade do Container.

Seguindo essa arquitetura foi implementado os módulos de login (Figura 5), registro diário (Figura 6), horário semanal (Figura 7) e férias (Figura 8). Para cada um dos módulos, uma View e um Container fora criado. Em Views complexas como a do horário diário, componentes React menores são criados, contribuindo para organização, facilitando a manutenção e possibilitando o reuso. Abaixo é detalhado o funcionamento e comportamento da aplicação passando desde como são feitas as requisições para a API até a exibição dessas informações para o usuário.

11.7.1 Implementação

As Views são alimentadas com dados fornecidos pela API REST e para consumi-los é necessário realizar requisições HTTP para os endpoints disponibilizados pela mesma. Para este fim foi utilizado a biblioteca JavaScript Axios, que implementa em alto nível as principais

necessidades de uma requisição HTTP, como tipo da requisição (POST, GET, etc), parâmetros e conteúdo do corpo da mesma. Com essa biblioteca foi então implementado para cada domínio do modelo, um objeto responsável por interagir com a API REST. Estes objetos, realizam, em sua grande maioria, operações de adição, edição, busca e deleção.

Implementado a comunicação com a API, é preciso de um mecanismo que gerencie essa interação, disponibilizando os dados recebidos para toda a aplicação, informando o status de cada requisição e lidando com requisições assíncronas. Essas tarefas são executadas pelo Requester, ferramenta da biblioteca Bridge-React. O Requester utiliza os objetos de comunicação com a API e a *store* da biblioteca Redux para gerenciar todo esse processo. Na *store* ficam armazenados todos os dados do modelo no lado do cliente. Operando sobre essas ferramentas o Requester consegue atender, com uma interface simples, as principais necessidades de comunicação e armazenamento da aplicação.

Para operar sobre o Requester e criar as Views fora criado o Container. Nele é implementado a lógica de negócio e as funções necessárias para o funcionamento da View. As Views recebem apenas os dados e as funções já implementadas pelo Container, ficando independente da lógica do domínio e focando apenas na implementação dos elementos gráficos.

Implementado em React as Views são compostas, em sua grande maioria, por outros componentes menores que juntos formam uma página complexa que representa um modelo. Muitos destes componentes, foram disponibilizados pela biblioteca Bridge React e adaptados para esse projeto. Quando necessário novos componentes foram criados para atender as necessidades específicas.

Buscando melhorar a usabilidade e responsividade da aplicação as operações sobre intervalo de tempo foram implementadas também no lado do cliente. Foram criadas classes utilitárias responsáveis por operar sobre um tipo especifico de dado, como, por exemplo, hora. Todas as operações sobre esse tipo de dado ficam centralizadas em um único objeto contribuindo para manutenção. Além disso, menos requisições para o servidor são feitas, já que a necessidade foi atendida no lado do cliente.

Seguindo a arquitetura apresentada foram implementadas todos os módulos propostos para esse trabalho, a segmentação e delegação de responsabilidades propostas facilita a evolução da aplicação permitindo que novos módulos sejam adicionados sem grandes dependências dos já existentes.

11.7.2 Login

Para realizar o controle de acesso da aplicação foi criado um componente de Login, que tem como função verificar se o usuário está autenticado ou não. Caso esteja o usuário é redirecionado para a URL desejada, caso contrário é redirecionado para a tela de login. O componente armazena na store do Redux os dados do usuário logado. No primeiro acesso a store estará vazia, então o componente realiza uma requisição para o endpoint */api/user/current* da API REST que irá retornar os dados do usuário logado ou erro 401 indicando que o mesmo não esta autenticado com o servidor.

Ao receber o erro 401, o componente redireciona o usuário para a tela de login, que o permite acessar o Servidor de Autorização do Google, responsável por conceder a autenticação. Após se autenticar, o servidor o redireciona para a URL inicial da aplicação. Dessa vez, o componente de login terá como resposta da API REST os dados do usuário, indicando que o mesmo está autenticado. Os dados do usuário então são adicionados na store e ficam disponíveis para qualquer componente da aplicação que precise dessas informações.

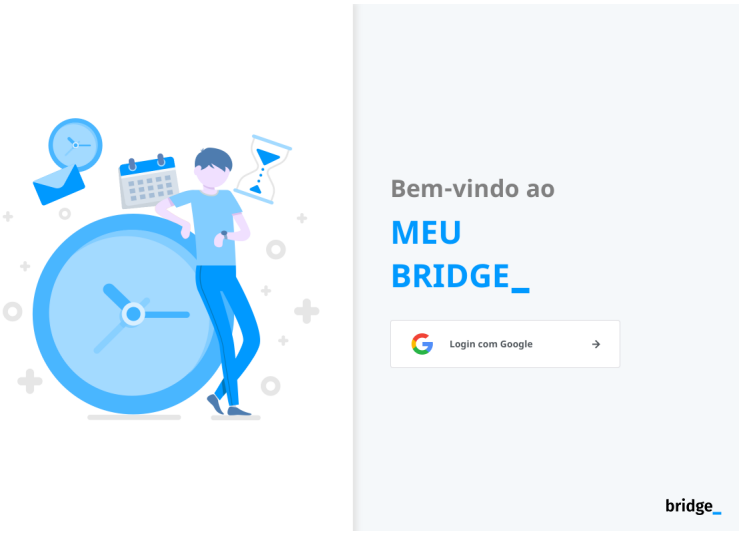

Figura 5 – Tela de Login

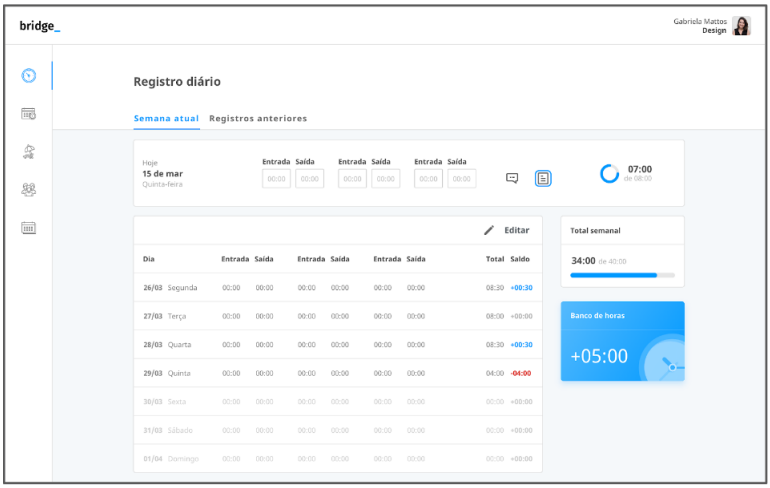

Figura 6 – Tela do módulo de Registro diário

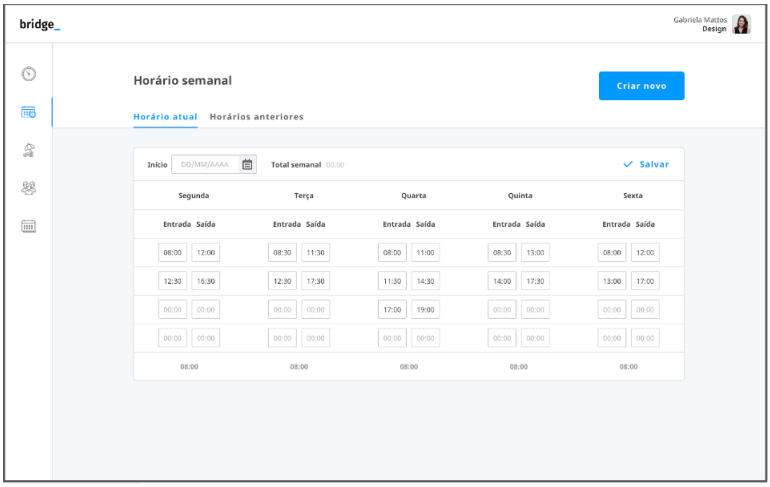

Figura 7 – Tela do módulo de Horário semanal

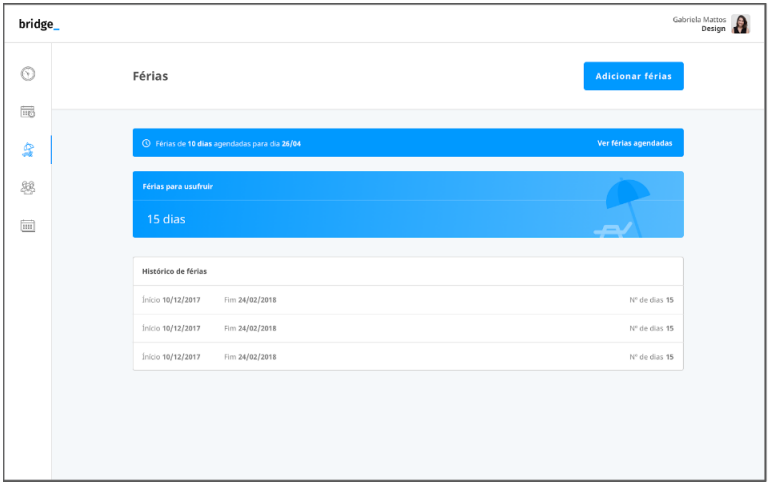

Figura 8 – Tela do módulo de Férias

12VALIDAÇÃO DA APLICAÇÃO

Para executar a validação do software foram tomadas duas iniciativas, uma delas ainda com o cliente e figuras de mais alto calão do laboratório e outra diretamente com os usuários finais da aplicação. A intenção fora fazer teste de aceitação sobre o produto implementado e posteriormente avaliar a experiência de uso das funcionalidades dentro de seu ambiente de produção.

12.1 TESTE DE ACEITAÇÃO

Em primeiro momento, ainda no decorrer dos ajustes finais da implementação, nos reunimos com o gerente administrativo do laboratório, qual assumira o papel de cliente deste projeto. Com nosso acompanhamento, em algumas iterações, esse foi colocado frente ao software para experimentar sua utilização e, conforme prevê uma etapa de teste de aceitação, avaliou se os serviços proporcionados pelo sistema atendiam às necessidades organizacionais para o qual este fora projetado.

Para essa avaliação também foram consultados demais lideres da organização, afim de trazer uma visão estratégica sobre a implantação da ferramenta e o que mais ela precisaria abranger para servir de apoio as politicas qual se relaciona.

A partir dessas interações foram realizados ajustes finais nas funcionalidades afim de complementar o até então desenvolvido para que atendesse às expectativas levantadas durante o teste de aceitação. Aquilo que fora dado como evolução, seguiu encaminhado para as futuras versões do sistema, conforme o gerenciamento de prazo e escopo estipulado para a entrega da aplicação e trabalho acadêmico.

Tabela 3 – Avaliação comparativa do Horário Semanal

Característica	Avaliação
Configuração inicial	90% julgou mais fácil
Facilidade de preenchimento	20% julgou mais difícil
Consulta aos registros anteriores	80% julgou mais fácil

Tabela 4 – Avaliação comparativa do Registro diário

Característica	Avaliação
Encontrar a data atual	100% julgou mais fácil
Visualizar a justificativa adicionada	30% não julgou mais fácil
Descobrir o dia da semana associado à data	40% julgou igual
Consultar o banco de horas	100% julgou mais fácil

Tabela 5 – Avaliação comparativa da gestão de férias

Característica	Avaliação
Visualizar os dias que tem direito	90% julgou mais fácil
Registrar um período de férias	100% julgou mais fácil
Identificar férias anteriores e planejadas	100% julgou mais fácil

12.2 TESTE DE USABILIDADE

Em sequência fora aplicado junto com os colaboradores do laboratório, efetivos usuários da aplicação, um teste de usabilidade para avaliarmos grandes gargalos que a proposta de interface ou os próprios fluxos previstos pudessem ter. A ideia foi colocar os usuários frente ao sistema e designá-los um roteiro de atividades para que, sem auxílio explicativo, acompanhássemos as dificuldades que passariam pra realizar as principais tarefas disponíveis pelo sistema.

Sabendo que os participantes tinham como parâmetro apenas o funcionamento da ferramenta utilizada até então, pudemos observar quão intuitivo estava a utilização de cada serviço, se esses traziam facilidade de uso em relação a proposta anterior e também sugestões sobre evoluções que poderiam agregar à ferramenta para que essa auxiliasse ainda mais as tarefas previstas.

Para essa dinâmica, coletamos a opinião de 10 voluntários, através de um questionário aplicado pelo Google Forms, qual consta anexado ao trabalho. Esse fora dividido em 3 partes. Na primeira destrinchamos características das 3 principais funcionalidades e pedimos que os avaliadores comparassem o ganho (ou perda) obtida com esta plataforma em relação a que eles estão acostumados a usar.

De modo geral as respostas indicaram benefícios na substituição da ferramenta, ainda assim houve o que, na visão daquela amostra, não trouxe tantas mudanças. Resumimos a seguir alguns dos pontos

avaliados e suas respostas nas tabelas 3, 4 e 5.

Ainda no mesmo questionário colocamos algumas perguntas sugeridas pelo System Usability Scale (SUS) que visa avaliar a satisfação do usuário em interagir com aquela aplicação. Esta proposta intercala questionamentos positivos e negativos para obter uma visão de quão confortável ou árduo fora o contato do participante com as tarefas realizadas no sistema.

Por fim, nosso processo avaliativo, abria espaço para que os envolvidos colocassem, em suas palavras, melhorias que as funcionalidades disponibilizadas poderiam sofrer para que auxiliassem ainda mais as tarefas as quais se propõe, bem como outros serviços que a aplicação poderia trazer para cooperar com as vivências organizacionais daqueles colaboradores.

As respostas de todas as etapas foram, em geral, positivas e constataram entusiasmo na utilização do produto tanto por parte da gerência quanto dos colaboradores. Essa coleta também permitiu-nos considerar o que se espera da evolução do sistema e assim ter material para guiar suas correções, melhorias e evoluções daqui por diante.

13CONCLUSÃO

Considerando as intenções estabelecidas na proposta desse projeto somado às experiências vivenciadas e construções viabilizadas durante sua execução, fica explicito o alcance do objetivo geral que propunha *implementar uma solução de software para a apoio à gestão do banco de horas e das férias dos colaboradores de uma instituição.*

A citar, conforme proposto, o desenvolvimento e disponibilização da API REST com arquitetura planejada para simplificar sua expansão, uma aplicação web a essa integrada, ambas desenvolvidas conforme processo de trabalho previsto e bem estruturado com forte participação do cliente e designer.

Não só isso, as funcionalidades que estavam especificadas foram desenvolvidas em sua totalidade, logo, o sistema disponibiliza os serviços para o qual fora concebido, e atende, portanto, às necessidades organizacionais que formaram as principais motivações por trás desse projeto.

Em complemento, como previsto, a efetividade do sistema construído foi avaliada junto com o cliente e usuários, indicando satisfação dos resultados obtidos e entusiasmo por sua aplicação como ferramenta de apoio às politicas do laboratório.

Como demonstrado, ainda, na validação do software, sua utilização tende a contribuir consideravelmente na usabilidade das tarefas relacionadas, tornando-se relevante para a vivência dos participantes da instituição.

Além do mais, o envolvimento de outros desenvolvedores com plataforma está planejada ainda para esse ano, garantindo sua consolidação como ambiente de propagação das tecnologias utilizadas.

Também é de muito valor considerar o aprendizado que adquirimos durante a execução do trabalho. A implementação realizada nos colocou a frente de diversos desafios, cuja solução demandou que elevássemos nosso conhecimento, fosse buscando apoio dos colegas ou em pesquisa de material relacionado.

É ainda pontuável a aplicação das teorias vistas ao longo do curso em diversos dos momentos percorridos nessa implementação, não só a contar as matérias primarias, mas em especial as disciplinas a respeito de banco de dados, engenharia de software e gerenciamento de projetos.

Em suma, o trabalho foi de extrema relevância, não só pelo produto proporcionado, mas também como componente importante na formação dos, em breve, cientistas da computação que o executaram.

14TRABALHOS FUTUROS

Surgindo da intenção de informatizar processos administrativos de uma organização, são inúmeros os serviços desejados a serem compreendidos pelo software proposto. Já em sua concepção eram esboçados uma série de funções em diferentes domínios que seriam bem vindas para um software dedicado à gestão de pessoas e processos administrativos de uma instituição.

Além do mais, por ter sido fundado dentro de um laboratório universitário, desde o início esse projeto teve como foco de sua construção o objetivo de servir como ambiente de desenvolvimento para o aprendizado das tecnologias utilizadas. E é previsto desde então que a plataforma siga sendo evoluída dentro da instituição de modo a introduzir outros colaboradores do Bridge às práticas e ferramentas, neste já empregadas, e que passarão a compor os processos de desenvolvimento dos demais softwares implementados na instituição.

Por conclusão, como essência de um projeto acadêmico e por ter sua codificação publicada com direitos de reutilização, as contribuições realizadas ficam a disposição da comunidade, possibilitando a qualquer interessado cooperar com seu desenvolvimento conforme necessidades identificadas por aqueles que se utilizarão da plataforma.

14.1 EVOLUÇÕES DO SOFTWARE

Tomando isso em vista, enumeramos a seguir algumas das evoluções estruturais, integrações e novas funcionalidades percebidas como relevantes para a utilização do software nas vivencias organizacionais do laboratório ou de instituições semelhantes:

- **Funcionalidades complementares**

    1. Pesquisa e acesso às informações de outros colaboradores:
       Centralizando ambiente de consulta ao horário de expediente e planejamento de férias de um colega de trabalho;
    2. Módulo para acesso ao calendário de expediente:
       Permitindo consulta centralizada aos feriados e eventos planejados pela instituição;
    3. Implementação das melhorias sugeridas na etapa de avaliação do sistema:

A citar, facilidade de preenchimento dos campos de hora e data e registro simplificado da hora atual;

4. Módulo de configuração dos feriados e eventos da instituição:
   Permitindo que essas informações sejam gerenciáveis através da própria aplicação.

- **Infraestrutura da aplicação**

1. Gerenciamento de perfis e recursos dos usuários:
   Permitindo configuração de acesso diferenciado às funcionalidades conforme papel organizacional;
2. Módulo de auditoria:
   Para registro das ações realizadas pelos usuários dentro do sistema;
3. Automatização de testes sobre as construções do front-end.

- **Integrações externas**

1. Enviar e-mail a partir de ações dos usuários:
   Otimizando ao processo administrativo referente à solicitação de férias de um colaborador, ou ainda a título de notificações de eventos ou processos pendentes desses.
2. Integração com o Google agenda:
   De modo a alimentar os horários de um usuário para utilização das funcionalidades e composição gráfica disponível pela ferramenta.

- **Outros serviços**

1. Módulo para divulgação de comunicados da organização:
   Centralizando a comunicação da instituição com seus colaboradores.
2. Módulo para gerenciamento de contratos de celetistas e bolsistas:
   Auxiliando os processos administrativos e judiciais referentes a essas burocracias.

Com isso pontuado, incentivamos iniciativas que visem investir no progresso e aplicação da ferramenta, nos disponibilizando para discussões referentes a essas investidas e intuindo colaborar como pudermos para auxiliar suas evoluções.

REFERĘNCIAS

AROUCK, O. Avaliação de sistemas de informação: revisão da literatura. *Nucleic acids research*, Transinformação, v. 13, n. 1, p. 7–21, 2001.

BIHIS, C. *Mastering OAuth 2.0*. [S.l.]: Packt Publishing Ltd, 2015.

Dan Abramov. *Motivation*. 2018. [Online; acessado 19 de novembro de 2018]. <https://redux.js.org/introduction/motivation>.

FIELDING, R. T. Rest : Architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000. <https://ci.nii.ac.jp/naid/20000913235/en/>.

FINK, G. et al. *Pro Single Page Application Development: Using Backbone. Js and ASP. Net*. [S.l.]: Apress, 2014.

GACKENHEIMER, C. What is react? In: *Introduction to React*. [S.l.]: Springer, 2015. p. 1–20.

Governo Federal Brasileiro. *Art. 59, § 2 da Consolidação das Leis do Trabalho, decreto de lei nº 5.452*. 1943. https://www.jusbrasil.com.br/topicos/10759744/paragrafo-2-artigo-59-do-decreto-lei-n-5452-de-01-de-maio-de-1943. Online; acessado 28 de Outubro de 2018.

HARDT, D. The oauth 2.0 authorization framework. 2012.

Kent Beck et al. *Manifesto for Agile Software Development*. 2001. http://www.manifestoagil.com.br/principios.html. Online; acessado 08 de novembro de 2018.

KRASNER, G. E.; POPE, S. T. et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, v. 1, n. 3, p. 26–49, 1988.

LEFF, A.; RAYFIELD, J. T. Web-application development using the model/view/controller design pattern. In: IEEE. *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*. [S.l.], 2001. p. 118–127.

LI, L.; CHOU, W. Design and describe rest api without violating rest: A petri net based approach. In: IEEE. *Web Services (ICWS), 2011 IEEE International Conference on*. [S.l.], 2011. p. 508–515.

MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2011.

MATTOS, G. Desenvolvimento de uma ferramenta digital para comunicação interna do laboratório bridge. 2017.

MYERS, G. J. *The Art of Software Testing*. [S.l.]: John Wiley Sons, 2004.

NBR, A. Iso 9001/2000: Sistemas de gestão da qualidade. 2001.

OCUPACIONAL, R. B. de S. Os novos paradigmas de organização do trabalho: implicações na saúde mental dos trabalhadores. v. 23, n. 85/86, 1997.

Open Api Initiative. *OpenAPI Specification*. 2017. [Online; acessado 14 de fevereiro de 2018]. <https://github.com/OAI/OpenAPI-Specification>.

PATNI, S. *Pro RESTful APIs*. [S.l.]: Springer, 2017.

Pete Hunt. *Why did we build React?* 2013. [Online; acessado 17 de novembro de 2018]. <https://reactjs.org/blog/2013/06/05/why-react.html>.

RTCA/EUROCAE. *Software Considerations in Airborne Systems and Equipment Certification*. [S.l.]: Washington, D.C., EUA, 1992.

Spring. *Spring Framework Overview*. 2018. [Online; acessado 17 de novembro de 2018]. <https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html>.

Spring Boot. *Spring Boot*. 2018. [Online; acessado 17 de novembro de 2018]. <https://spring.io/projects/spring-boot>.

Spring Security. *Spring Security*. 2018. [Online; acessado 17 de novembro de 2018]. <https://spring.io/projects/spring-security>.

Spring Web MVC. *Web on Servlet Stack*. 2018. [Online; acessado 17 de novembro de 2018]. <https://docs.spring.io/spring/docs/current/spring-framework-reference/web.htmlmvc>.

YATES, A. et al. The ensembl rest api: Ensembl data for any language. *Bioinformatics*, Oxford University Press, v. 31, n. 1, p. 143–145, 2014.