UNIVERSIDADE FEDERAL DE SANTA CATARINA

SWAP:

um sistema web para inscrição e auxílio às atividades do Programa de Pós-Graduação em Ciência da Computação

Daniel Blank

Florianópolis – SC
2011/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Sistema web para auxiliar os processos de alunos e professores no Programa de Pós-Graduação em Ciência da Computação

Daniel Blank

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis – SC
2011/2

Daniel Blank

Sistema web para auxiliar os processos de alunos e professores no Programa de Pós-Graduação em Ciência da Computação

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadora: _____
Profa. Dra. Carina Friedrich Dorneles

Banca examinadora

_____
Prof. Dr. Mário Antônio Ribeiro Dantas

_____
Prof. Dr. Ronaldo dos Santos Mello

Universidade Federal de Santa Catarina

2011/2

Sumário

Lista de Figuras

Lista de Reduções

- PPGCC - Programa de Pós-Graduação em Ciência da Computação
- UFSC - Universidade Federal de Santa Catarina
- INE - Departamento de Informática e Estatística
- MVC - Model View Controller
- IDE - Integrated Development Environment
- JSF - JavaServer Faces
- POSCOMP - Exame Nacional para Ingresso na Pós-Graduação em Computação
- ORM - Object-Relational Mapping
- POJO - Plain Old Java Objects
- DAO - Data Access Object
- XML - Extensible Markup Language
- SeTIC - Superintendência de Governança Eletrônica e Tecnologia da Informação e Comunicação
- TCC - Trabalho de Conclusão de Curso

Resumo

Este trabalho tem como objetivo o desenvolvimento de uma aplicação web para auxiliar os alunos na inscrição ao mestrado no Programa de Pós-Graduação em Ciência da Computação (PPGCC), bem com auxiliar alunos e professores nas atividades do programa.

Através de um sistema de recomendação, a aplicação é capaz de recomendar quais os professores são mais indicados para o aluno na hora da sua inscrição, dependendo das áreas de interesses de cada um. Após as inscrições dos alunos, os professores também poderão solicitar recomendações, auxiliando na escolha de alunos para orientar.

Outras funcionalidades que a aplicação visa auxiliar são os processos de composição das bancas de mestrado e a imposição dos limites de alunos que um professor pode orientar. Com o intuito de facilitar e agilizar o trabalho dos responsáveis nestes processos, acabando com as partes burocráticas, trocas sucessivas de e-mails e organização das informações em documentos impressos, estes processos foram totalmente automatizados.

Palavras-chave: Java. JSF. PrimeFaces. Spring. Hibernate. MVC. PostgreSQL. Banco de Dados. Sistema de Recomendação. Pós-Graduação. Ciência da Computação.

1. Introdução

Para inscrever-se no mestrado do PPGCC, o candidato deve ser portador de diploma de nível superior compatível com a área de interesse, além de ter que realizar o "Exame Nacional para Ingresso na Pós-Graduação em Computação" (POSCOMP). Para realizar a inscrição no processo seletivo do Programa, os candidatos devem preencher a Ficha de Inscrição disponível no site do Sistema de Controle Acadêmico da Pós-Graduação (http://www.capg.ufsc.br/inscricao). Durante a inscrição no sistema o aluno deve escolher um professor para orientá-lo durante o período do mestrado, porém, se não for escolhido nenhum professor, cabe a algum professor escolhê-lo após o processo de inscrição. As bancas de mestrado só podem ser formadas por professores doutores. Diferentemente do processo de mestrado, o ingresso na graduação requer o ensino médio completo e a realização da prova do vestibular. Após isso, se o aluno for aprovado, é necessário apenas realizar a matrícula na universidade.

O processo de seleção de novos alunos para ingresso no PPGCC, em nível de mestrado, é baseado no número de vagas disponível para cada professor de cada linha de pesquisa. O número de vagas é determinado de acordo com uma classificação previamente efetuada pela Comissão de Produção Científica - CPC.

Para o aluno descobrir qual o professor mais adequado ao seu plano de trabalho no mestrado, é necessário que ele pesquise por este professor no site do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da

UFSC[1]. O caso do professor escolher alunos é mais complicado, pois é necessário que ele vá à secretaria do curso e verifique nos documentos de inscrição impressos de cada aluno se algum se encaixa na sua linha de pesquisa. Ambos os processos implicam em tempo disponível e paciência, principalmente no caso do professor. Para incluir um membro na banca de mestrado, é necessário comunicar ao coordenador do programa para junto com uma comissão avaliar se o membro está apto ou não a fazer parte da banca. Este processo é burocrático e pode demandar algum tempo para ser concretizado. Os professores têm um determinado limite de orientações, o qual é controlado através de papéis, tornando o controle mais difícil e suscetível a falhas por parte do responsável.

O preenchimento das inscrições dos candidatos não é feita em um site administrado pelo INE. Os dados requeridos para a inscrição são restritos comparado ao ideal para o INE, além de dificultar o acesso e a manipulação dos dados dos candidatos. No sistema atual não é possível visualizar a nota do aluno no POSCOMP e nem identificar se um aluno entrou em contato com algum professor previamente.

Todo o processo de definição de qual professor orientará quais alunos e o limite de alunos por professor, além do processo de elaboração das bancas de mestrado e aceitação dos professores nelas, são realizados em papel ou por trocas de e-mails entre os envolvidos. Estes processos acarretam em diversos problemas, como a dificuldade do responsável em organizar e administrar quais alunos já tem orientador definido, quantos alunos um professor ainda poderá orientar, a composição formalizada de cada banca, etc.

[1] www.ppgcc.inf.ufsc.br

Devido a grande dificuldade das pessoas em tomar decisões quanto à escolha de uma grande diversidade de itens e recursos ou da falta de conhecimento das opções, foram propostos os sistemas de recomendações. O objetivo destes sistemas é auxiliar o usuário na escolha do que ele deseja. Sem este auxílio o usuário poderia levar muito tempo até encontrar o que procura, ou até não encontrar. Os sistemas de recomendações visam filtrar informações que possam ser de interesse do usuário com base nas suas informações e características, e a partir delas recomendar-lhe itens e recursos.

Objetivos

Desta forma, propõe-se o desenvolvimento de um sistema de recomendação com o objetivo de auxiliar os alunos na escolha de um professor para orientação, recomendando professores que mais se assemelham ao plano de trabalho do aluno, sem que este precise analisar professor por professor até encontrar o mais adequado. Para os professores, o sistema auxilia recomendando alunos que mais combinam com suas linhas de pesquisa e projetos, evitando assim que o professor necessite procurar exaustivamente por alunos em documentos impressos. Outra situação que o sistema visa eliminar é o processo trabalhoso do responsável na gestão das bancas. Para evitar o uso de papéis e múltiplas trocas de e-mails, o sistema propõe um processo automatizado para definir as bancas de uma forma mais rápida e eficiente para os envolvidos neste processo.

Os objetivos específicos deste trabalho são:

- Auxiliar o candidato na inscrição no Programa de Pós-Graduação em Ciência da Computação (PPGCC);
- Auxiliar o aluno na escolha de um orientador, através de recomendação;
- Auxiliar o professor na escolha de um aluno para orientar, através de recomendação;
- Simplificar o processo de definição das bancas de mestrado.

Motivação

A principal motivação para a realização deste trabalho teve como intuito auxiliar no processo de inscrição do aluno, pois o mesmo pode não saber qual área seguir ou qual professor se adequa mais com os seus interesses. Neste caso, o aluno pode levar um tempo até analisar as linhas de pesquisa e os professores para tomar sua decisão. Após o processo de inscrição, os professores podem analisar os alunos que ainda não contataram algum professor, o que pode levar certo tempo também, já que é preciso analisar os interesses de cada um.

O processo de composição das bancas de mestrado demanda muito trabalho ao responsável, pois ele precisa lidar com diversas solicitações de inclusão de membros em bancas, tendo que realizar sucessivas trocas de e-mails. O processo da imposição dos limites de alunos que um professor pode orientar é realizado com documentos impressos, tornando o controle e organização dos documentos complicado, podendo perdê-los ou levar muito tempo até encontrar qual está procurando.

Este trabalho é organizado da seguinte forma. No Capítulo 2 é apresentada uma visão geral dos sistemas de recomendações, descrevendo as formas de realizar recomendações, os tipos de coleta de informações e as formas de disponibilização dos itens. O Capítulo 3 apresenta aplicações que utilizam sistemas de recomendações e o sistema web para suporte à coordenação de projetos do INE. A apresentação do sistema proposto é realizada no Capítulo 4, mostrando um mapa arquitetural do funcionamento do sistema e a descrição geral dividida em diversos módulos. No Capítulo 5 é abordada a forma de implementação do sistema, mostrando as tecnologias utilizadas para o desenvolvimento, além da descrição técnica do desenvolvimento de cada módulo. Finalmente, no Capítulo 6 são descritas as conclusões referentes a este trabalho, descrevendo os aprendizados que este trouxe, além das sugestões para trabalhos futuros a serem desenvolvidos como complemento para este trabalho.

2. Sistemas de Recomendação

Sistemas de Recomendação são classificados em três grupos de acordo com a forma na qual ele realiza as recomendações. O primeiro, denominado de Recomendações Baseadas em Conteúdo, baseia-se na recomendação de itens semelhantes aos já desejados pelo usuário, ou seja, analisa as semelhanças entre seus itens desejados com os demais itens disponíveis. Os interesses dos usuários são diferenciados, recomendando itens individualmente para cada usuário. O segundo grupo, denominado de Recomendações Colaborativas, baseia-se na recomendação de itens que foram desejados por usuários com interesses semelhantes ao usuário que realiza a consulta. Os interesses dos usuários são agrupados, criando comunidades de usuários que possuem interesses semelhantes. Este grupo se difere do primeiro por não exigir o reconhecimento do conteúdo dos itens. O último grupo de sistemas de recomendações, conhecido como Métodos Híbridos, aborda a combinação dos dois grupos citados anteriormente, com o objetivo de construir um sistema mais eficiente.

A coleta das informações referentes a um usuário pode ser realizada de forma explícita ou implícita. Na forma explícita, o usuário especifica seus interesses através do preenchimento de algum formulário. Já na forma implícita, seus interesses são coletados sem sua percepção, pois a coleta é realizada com base nos itens visitados e desejados por ele.

As recomendações de itens e usuários são disponibilizadas de acordo com a forma que são apresentadas no sistema, sendo dividido em três formas. A primeira denominada Push, consiste em recomendar itens ao usuário sem que

ele esteja interagindo com o sistema, como por exemplo, enviar e-mail ao usuário com recomendações de ofertas. A segunda, conhecida como Pull ocorre apenas quando o usuário solicita tal recomendação, possibilitando o controle de quando serão exibidas. A última forma de apresentação é denominada Passiva, onde as recomendações são exibidas no contexto do sistema, como por exemplo, itens mais vendidos de uma loja.

A seguir será apresentado como os sistemas de recomendação são classificados. Primeiramente será abordado as diferentes formas de como são realizadas as recomendações, em seguida, as formas de coletar informações do usuário, e finalmente as diferentes maneiras de disponibilizar as recomendações ao usuário.

2.1. Formas de recomendação

Baseado em Conteúdo

Este tipo de recomendação procura analisar o conteúdo de um item com o perfil ou interesse de um usuário, tendo como objetivo identificar se um item é relevante ao usuário. As informações referentes ao perfil ou interesse do usuário podem ser adquiridas pelo próprio usuário ou pelo seu comportamento no sistema, como construir seu perfil baseado em itens visualizados e desejados.

Outra forma de recomendação baseada em conteúdo é obter do usuário itens que ele considera de seu interesse ou não. Com base nestas informações

o sistema releva os itens semelhantes aos de interesse do usuário e descarta os semelhantes aos de não interesse.

Um dos problemas decorrentes deste tipo de sistema é quando há a inclusão de novas categorias de itens. Neste caso, o sistema não é capaz de recomendar estes itens com base no que o usuário visualizou ou desejou no passado. Outro problema é a ausência de surpresas na recomendação, ou seja, itens que não se assemelham com o perfil do usuário podem nunca ser recomendados, deixando de recomendar itens que poderiam ser de interesse do usuário. A recomendação pode ser prejudicada devido ao uso de sinônimos nas descrições de itens e perfis de usuários, deixando de recomendar itens que pudessem ser de interesse do usuário.

Recomendação Colaborativa

A recomendação colaborativa não necessita do conteúdo dos itens para realizar recomendações, pois é baseada nos interesses em comum entre os usuários, onde a correlação entre eles pode ocorrer item a item, como por exemplo, pessoas que compraram determinado produto também compraram outro mesmo produto em comum, ou correlação usuário a usuário, onde recomenda itens com base no que usuários com características semelhantes já desejaram. Esta forma de recomendação procura formar grupos de usuários com características semelhantes, onde a ideia principal está no compartilhamento de experiências entre os usuários.

A recomendação colaborativa possibilita recomendações inesperadas aos usuários, ou seja, o usuário pode receber recomendações de itens que estão fora do escopo de sua pesquisa.

Esta forma de recomendação pode apresentar algumas limitações. Quando um novo usuário é inserido no sistema, este necessita realizar diversas avaliações para se adequar a um grupo e assim poder receber recomendações. Quando um novo item é cadastrado no sistema, este não poderá ser recomendado para algum usuário devido à falta de avaliações de usuários. Quando o número de itens no sistema é muito grande em relação ao número de usuários, as avaliações ficam espalhadas pelos itens, dificultando o agrupamento de usuários similares. Caso um usuário tenha características distintas do normal, este poderá não encontrar um grupo de usuário com interesses similares, deixando a recomendação pouco eficiente.

Métodos Híbridos

Esta forma de recomendação combina a recomendação baseada em conteúdo e a recomendação colaborativa com o intuito de resolver os problemas encontrados em cada um para efetuar as recomendações com mais eficiência. Através da recomendação baseada em conteúdo obtêm-se boas recomendações para usuários com características distintas, ou seja, é possível considerar itens não avaliados por outros usuários e a eficiência indefere do número de usuários. Com a recomendação colaborativa há a possibilidade de obter recomendações inesperadas, além das avaliações de outros usuários serem aproveitadas para melhorar na eficiência das recomendações.

Na seção subsequente será abordado as formas que o sistema realiza a coleta das informações dos usuários, para assim poder efetuar recomendações de itens.

2.2. Coleta de informação

Explícita

Neste tipo de coleta de informação cabe ao usuário alimentar o sistema de forma voluntária. As informações são coletadas a partir do preenchimento de formulários a respeito do usuário, indicando suas preferências de itens que lhe interessam ou não. Além disso, o usuário pode avaliar itens do sistema, atribuindo notas e comentários.

A vantagem deste tipo de coleta é a confiabilidade das informações fornecidas, pois é o usuário que informa espontaneamente o seu perfil e o que é de seu interesse.

Uma das desvantagens da coleta explícita de informações é requerer uma maior disponibilidade de tempo do usuário, devido à necessidade de preenchimento de formulários e fornecimento de informações relevantes ao sistema. Alguns sistemas necessitam de formulários extensos para “conhecerem” melhor o usuário, exigindo paciência e atenção do usuário.

Implícita

Na coleta implícita as informações são obtidas sem a percepção do usuário. Estas são extraídas através de ações do usuário no sistema, como por

exemplo, aquisição e visualização de itens, páginas acessadas, realização de busca de itens, entre outras. A partir destas ações é possível conhecer o perfil do usuário e recomendar itens de seu interesse.

Neste tipo de coleta não é necessário que o usuário perca tempo com o preenchimento de formulários para o sistema obter informações a respeito do seu perfil. Em contrapartida, esta forma de coleta pode obter informações equivocadas em relação aos interesses do usuário. Por exemplo, o usuário pode estar pesquisando um item para outra pessoa, ou a conta de usuário pode ser acessada por mais de uma pessoa.

Na seção subsequente será abordado as formas que o sistema disponibiliza as recomendações ao usuário.

2.3. Disponibilização dos itens recomendados

Push

Esta forma de disponibilizar as recomendações ocorre sem uma requisição do usuário e que este nem esteja interagindo com o sistema. Um exemplo desta forma é o envio de e-mail ao usuário com recomendações, o que pode ser considerado uma vantagem, pois o usuário não precisa entrar constantemente no sistema para procurar itens que lhe interessam, deixando a cargo do sistema lhe oferecer itens novos cadastrados no sistema que possam ser de interesse do usuário.

Pull

Nesta forma de disponibilizar as recomendações, cabe ao usuário decidir quando elas serão exibidas. A vantagem é que o usuário tem o controle de quando elas serão mostradas. Normalmente há um botão no sistema para o usuário clicar e serem apresentadas as recomendações a ele.

Passiva

Na forma passiva as recomendações são apresentadas ao usuário no contexto do sistema, sem nenhum aviso e sem que ele possa controlar tais recomendações. Aparecem como itens mais vendidos ou itens semelhantes aos adquiridos e visualizados pelo usuário no sistema.

3. Trabalhos Relacionados

Este capítulo apresenta duas categorias de trabalhos: sistemas de recomendação desenvolvidos e disponibilizados na web para uso geral e um sistema de auxílio no processo de avaliação e montagem de banca para trabalhos de conclusão de curso.

3.1. Recomendação de produtos a pessoas

Nesta seção são descritos alguns sistemas de recomendações que recomendam produtos a pessoas, como por exemplo, livros, filmes, músicas, etc.

Primeiramente é apresentado o sistema MovieLens[2], um sistema de recomendação desenvolvido por um laboratório de pesquisa da Universidade de Minnesota. Este sistema utiliza a técnica de filtragem colaborativa para recomendar filmes para seus usuários assistirem.

Para usufruir das recomendações de filmes é necessário realizar um cadastro simples no site. Após a realização do cadastro o sistema precisa que o usuário avalie quinze filmes. Para isto o sistema lista diversos filmes ao usuário até que atinja quinze avaliações. A partir deste momento o sistema está apto a realizar recomendações de filmes que provavelmente o usuário irá gostar de assistir, como mostrado na Figura 1. Vale ressaltar que quanto mais filmes o usuário avaliar mais precisas serão as recomendações.

[2] www.movielens.org

Figura 1 – Tela do site do MovieLens onde são recomendados filmes aos usuários

O sistema compara as avaliações de um usuário com as avaliações de outros usuários com gosto semelhante aos seus. Através das análises das avaliações o sistema guarda informações, tal como, se os usuários *x* e *y* avaliaram positivamente um mesmo filme, os filmes avaliados positivamente pelo usuário *y* podem ser recomendados ao usuário *x*. A partir dessas informações e da coleta explícita das avaliações do usuário é recomendado em uma lista ordenada os filmes que provavelmente irá mais lhe agradar.

Outro exemplo de sistema de recomendação de itens a usuários é o sistema Gnod[3], o qual é dividido em três subsistemas, Gnod Music[4], Gnod Books[5] e Gnod Movies[6], que recomendam respectivamente, artistas musicais, autores e filmes.

[3] www.gnod.net
[4] www.gnoosic.com
[5] www.gnooks.com
[6] www.gnovies.com

Como o funcionamento dos três subsistemas é idêntico, será detalhado e exemplificado apenas um deles, o Gnod Music. Para obter os dados necessários para realização das recomendações, o sistema utiliza a coleta explícita de informações, solicitando três artistas musicais preferidos do usuário, como visto na Figura 2.

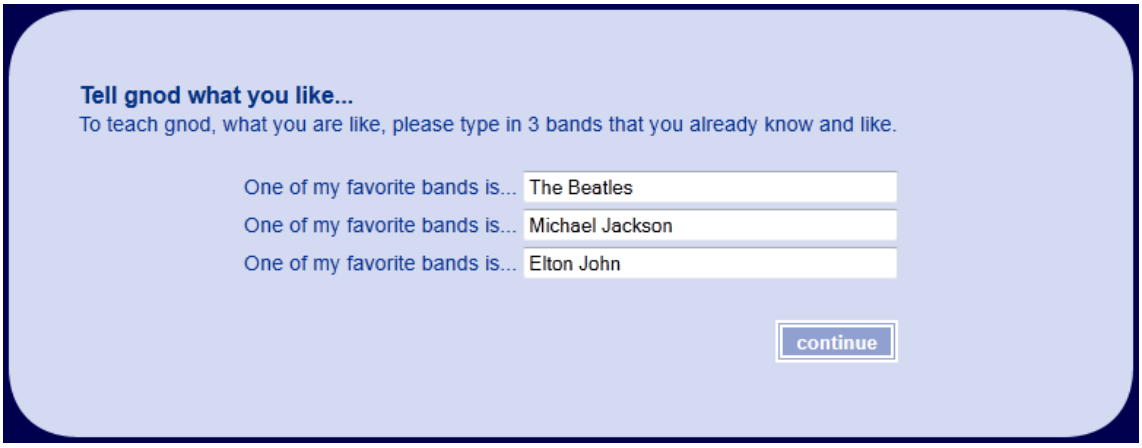

Figura 2 - Site da Gnod Music, onde são inseridos os artistas preferidos.

De acordo com os dados enviados, o sistema recomendará artistas que se assemelham ao seu gosto musical, como mostrado na Figura 3. Este sistema utiliza a técnica de filtragem baseada em conteúdo para recomendar os artistas ao usuário.

Figura 3 - Site da Gnod Music, onde são recomendados artistas.

3.2. Recomendação de pessoas a pessoas

Esta seção apresenta alguns sistemas de recomendação que recomendam pessoas ao usuário do sistema baseando-se em interesses, perfis ou outras características em comum.

O sistema web eHarmony[7] é um site de relacionamentos fundado em 2000 nos Estados Unidos e lançado no ano de 2010 no Brasil. Os princípios do sistema foram desenvolvidos por um psicólogo que percebeu que o motivo dos casamentos durarem pouco ocorria devido a certas diferenças entre os casais. Sabendo disso, ele elaborou um esquema para relacionar pessoas que combinam características com grandes chances de um relacionamento bem sucedido.

O eHarmony se diferencia dos outros sites de relacionamento pelo fato de não precisar procurar por pessoas com perfis semelhantes aos seus entre os milhares existentes. Com o preenchimento de um vasto formulário a respeito

[7] www.eharmony.com.br

de suas características, o sistema irá fazer uma combinação com as características dos outros usuários, indicando assim pessoas que possam formar um relacionamento duradouro com você, conforme a Figura 4.

Após a indicação de pessoas, o usuário pode visualizar o perfil de uma delas, enviar perguntas pré-definidas e se comunicar via eHarmony Mail. Para utilizar o serviço completo e se comunicar livremente com outras pessoas, é necessário aderir aos planos pagos.

Figura 4 - Site da eHarmony onde são recomendadas pessoas que combinam com um usuário

O eHarmony é um sistema de recomendação de pessoas a pessoas que se baseia em pesquisas científicas para determinar as características entre as pessoas que dariam certo em um relacionamento bem sucedido. A técnica de recomendação utilizada é a filtragem baseada em conteúdo. Nesse caso, o conteúdo seriam as características das pessoas. O que é analisado e considerado são as características semelhantes entre as pessoas. A coleta de informações é explícita, pois toda a informação referente a uma pessoa é ela própria que preenche no sistema.

Outro sistema que recomenda pessoas ao usuário é o Facebook[8], que possui uma função de recomendação de amigos que utiliza as informações do perfil do usuário, como cidade natal, cidade em que mora, escola e faculdade onde estudou e local onde trabalha, para recomendar usuários. A coleta das informações do perfil do usuário é realizada de forma explícita através do preenchimento de um formulário. O Facebook utiliza a técnica de recomendação de filtragem baseada em conteúdo, onde compara o perfil de um usuário com os demais, sugerindo pessoas que possam ser conhecidas por ele. Outra forma de recomendação de amigos que o Facebook utiliza é sugerir ao usuário pessoas que seus amigos têm em comum, conforme a Figura 5. Por exemplo, quanto mais amigos do usuário uma pessoa conhece, maior a possibilidade de esta pessoa ser conhecida por ele. Além do Facebook, outros sistemas de rede social como, Orkut[9], Google+[10] e LinkedIn[11], também possuem um sistema de recomendação para recomendar usuários.

[8] www.facebook.com
[9] www.orkut.com
[10] plus.google.com
[11] www.linkedin.com

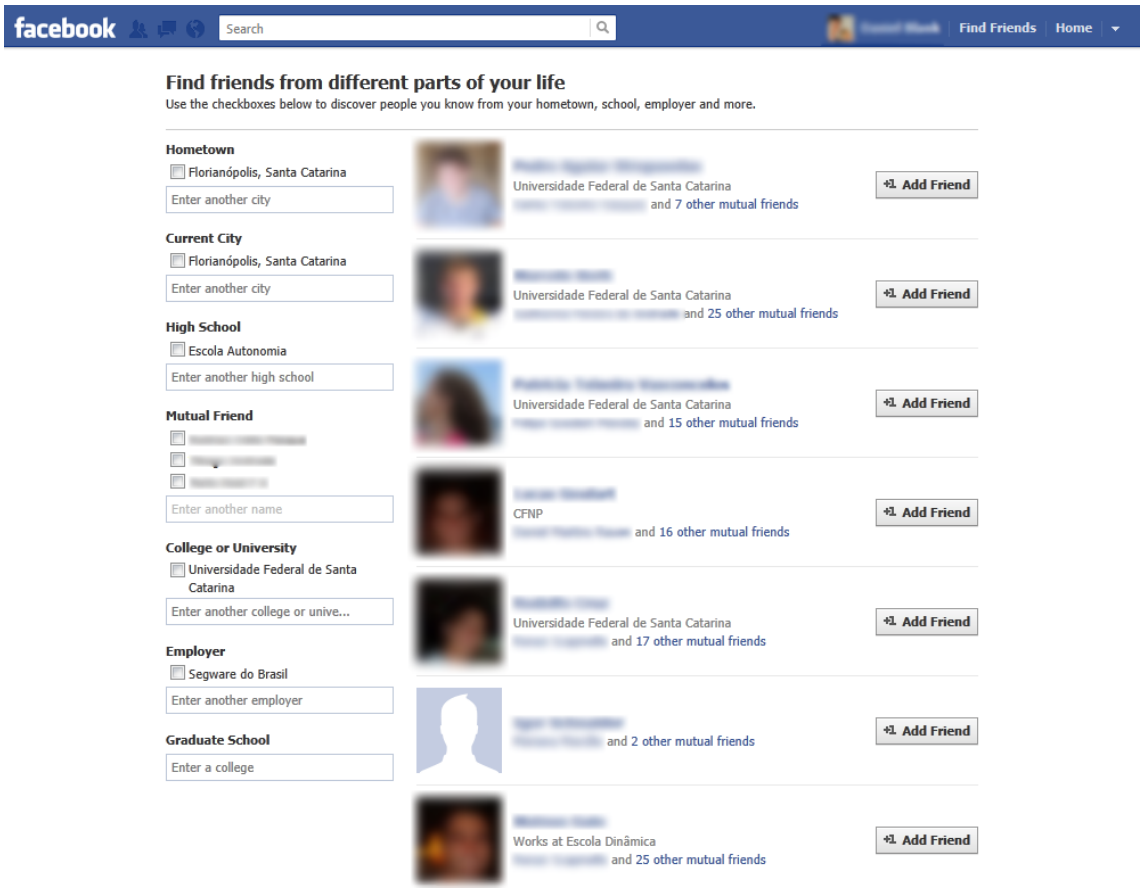

Figura 5 - Página que recomenda amigos no facebook

3.3. Avaliação e definição de banca de TCCs

Para a conclusão dos cursos em Ciências da Computação e Sistemas de Informação da Universidade Federal de Santa Catarina, é necessário o desenvolvimento do trabalho de conclusão de curso para aquisição do diploma. O currículo do curso disponibiliza três disciplinas, que tem por objetivo orientar o acadêmico no desenvolvimento do seu projeto e ainda pode avaliá-lo. As disciplinas são Introdução ao Projeto em Ciência da Computação, Projeto em Ciência da Computação 1 e Projeto em Ciência da Computação 2. Estas disciplinas requerem uma maior interação entre alunos, orientadores, coordenadores e membros de banca.

Os TCCs são desenvolvidos através da coordenação de um professor e do coordenador de projetos, que estipula prazos, define orientadores e avaliadores, entrega de relatórios, datas de apresentação dos projetos para os demais acadêmicos, etc.

Os processos envolvidos no desenvolvimento de um TCC eram estritamente burocráticos, pois eram realizados através do preenchimento de formulários em papel, o que exigia uma maior demanda. Para que este processo se tornasse menos burocrático foi desenvolvido o sistema web de Coordenação de Projetos[12], que tem por objetivo facilitar a obtenção de informações de projetos, melhorar a comunicação entre alunos e professores, permitir o preenchimento de formulários on-line, acompanhamento do estágio de desenvolvimento de um projeto e avaliação e definição de bancas de TCCs.

Com este sistema, a interação entre o aluno, orientador e membros da banca foi significativamente melhorado. Através deste, o processo de definição de banca foi simplificado, o orientador envia um convite a um professor, este é notificado e decide se aceita ou não em participar da banca. Com a banca formada, os alunos enviam documentos referentes à avaliação das disciplinas, os membros da banca e o orientador são notificados e têm acesso aos documentos no sistema. Após avaliarem os documentos, as notas são inseridas no sistema para o coordenador de projetos analisar e fechar as notas finais das disciplinas.

As dificuldades encontradas puderam ser eliminadas com a criação do sistema, onde há o controle das informações de alunos, orientadores e dos

[12] projetos.inf.ufsc.br

membros de banca, e ainda permite o controle de prazos e emite avisos quando necessário.

Através da análise dos sistemas de recomendação citados neste capítulo, percebemos a importância que eles têm para agilizar e auxiliar usuários na escolha de itens que não lhe são familiares. Devido a isto, foi possível compreender e implantar um sistema de recomendação de forma simples e eficiente.

4. SWAP

Este capítulo aborda a proposta de um sistema de recomendação que possui dois focos principais: (i) auxiliar alunos e professores no processo de inscrição no PPGCC através de um sistema de recomendação; e (ii) gerir as bancas de mestrado de uma forma prática e simplificar os processos de comunicação entre as partes envolvidas.

A forma de recomendação utilizada no desenvolvimento do sistema é a recomendação baseada em conteúdo pelo motivo de ser a mais adequada ao sistema proposto, pois como é uma relação direta entre aluno e professor, não se faz necessário criar grupos de usuários afins, mas sim comparar diretamente os conteúdos de alunos com os de professores e vice-versa.

O sistema utiliza a coleta explícita de informação para realizar as recomendações, pois como são utilizadas informações pessoais, o sistema não conseguiria extrair estas informações. Logo é necessário que o usuário insira explicitamente as informações no sistema.

A disponibilidade das recomendações é feita utilizando a técnica *pull*, para assim fornecer ao usuário o controle de quando as recomendações são realizadas.

4.1. Arquitetura

A Figura 6 apresenta o funcionamento de toda a aplicação, demonstrando esquematicamente os fluxos dos processos e a integração entre os módulos presentes no sistema. Nesta figura é possível ver os módulos presentes na

aplicação e a interação entre eles. O módulo de segurança, responsável pela autenticação do usuário no sistema, se comunica diretamente com o banco de dados para conferir se os dados envidados pelo usuário são válidos. O gerenciador de entidades é responsável por realizar as operações das entidades com o banco de dados, como salvar, atualizar, excluir e pesquisar por uma entidade. O módulo de inscrição é encarregado pelo processo de inscrição do aluno, sendo que no final deste processo o aluno é inserido no banco de dados. Ainda neste módulo é possível solicitar recomendações de professores pelo módulo casador de perfis. Outro módulo que utiliza o casador de perfis é o gerenciador de orientações, neste caso recomendando alunos ao professor que solicita pelo serviço. O gerenciador de orientações lida com o processo de escolha de alunos para orientação pelos professores e com o limite de orientações imposto pelo grupo de cada professor. O gerenciador de bancas é responsável por todo o processo de composição de uma banca de mestrado. O processo consiste na solicitação de inclusão de um membro na banca, seguido pela análise de um comitê de bancas e por fim a análise do professor convidado. Os módulos mais importantes deste sistema são o gerenciador de bancas e o casador de perfis.

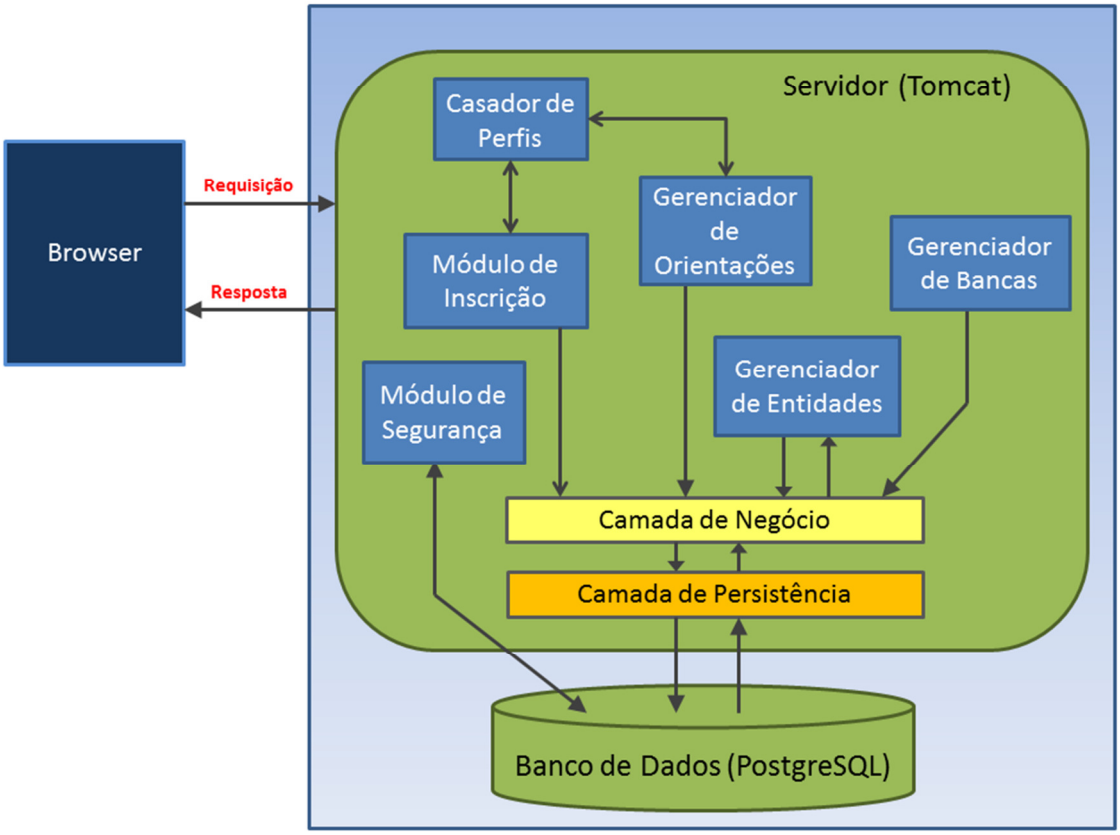

Figura 6 - Mapa Arquitetural da Aplicação

4.2. Casador de Perfis

Módulo responsável por auxiliar alunos na escolha do orientador com mais afinidade ao seu plano de trabalho e para professores na escolha de alunos que se encaixam com os seus projetos e linha de pesquisa.

Para os alunos, este módulo está presente durante o seu processo de inscrição. O aluno informa o seu plano de trabalho para o mestrado e na etapa seguinte do processo de sua inscrição, (onde ele seleciona a linha de pesquisa e o professor para possível orientação), há a opção de solicitar recomendações de professores. O sistema de recomendação se baseia na descrição do plano de trabalho informado pelo aluno com os objetivos das linhas de pesquisa e

com as descrições dos projetos de cada professor. Considerando esta análise o sistema lista os professores melhores classificados. A classificação é realizada através da média de scores do plano de trabalho com as linhas de pesquisas e projetos utilizando a função de similaridade Q-Grams Distance, que gera um score de zero até um, sendo que quanto mais próximo de um, mais similares são os textos comparados. A Figura 7 apresenta a página no sistema onde são mostrados os professores recomendados, onde o aluno tem acesso às linhas de pesquisa e projetos de cada professor recomendado, auxiliando assim o aluno na escolha de um orientador.

Figura 7 - Recomendação de professores no processo de inscrição do aluno.

Para os professores, este módulo está presente durante o processo de escolha de alunos para orientação. O professor solicita recomendações de alunos e com base nas linhas de pesquisa e dos projetos do professor, o

sistema analisa a similaridade com o plano de trabalho dos alunos que durante o processo de inscrição selecionaram uma linha de pesquisa do professor e/ou o próprio professor. A partir desta análise, o sistema classifica os alunos com base na média do score obtido na análise, usando a mesma função de similaridade e processo de análise descritos anteriormente na recomendação de professores a alunos. A Figura 8 mostra a área no sistema onde o professor seleciona alunos para orientação. Há a opção de solicitar a recomendação de alunos que mais se assemelham com suas linhas de pesquisa e projetos.

Figura 8 - Recomendação de alunos no processo de escolha de orientandos.

4.3. Gerenciador de Bancas

Este módulo é responsável pela gestão das bancas de defesa de mestrado, onde cabe a um comitê avaliar se um professor está apto a participar de uma defesa.

Os orientadores efetuam no sistema uma solicitação para inclusão de cada membro em uma banca (Figura 9). O presidente do Comitê de bancas, designado a avaliar as solicitações, é notificado e em seguida avalia junto ao comitê se o professor está apto ou não a participar da banca (Figura 10). Caso o professor seja aceito na banca, este é notificado sobre a sua participação e decide se possui interesse em participar ou não (Figura 11). O orientador é notificado de qualquer decisão ocorrida neste processo, ou seja, se o professor foi ou não aceito pelo comitê e se o professor aceitou ou não participar da banca.

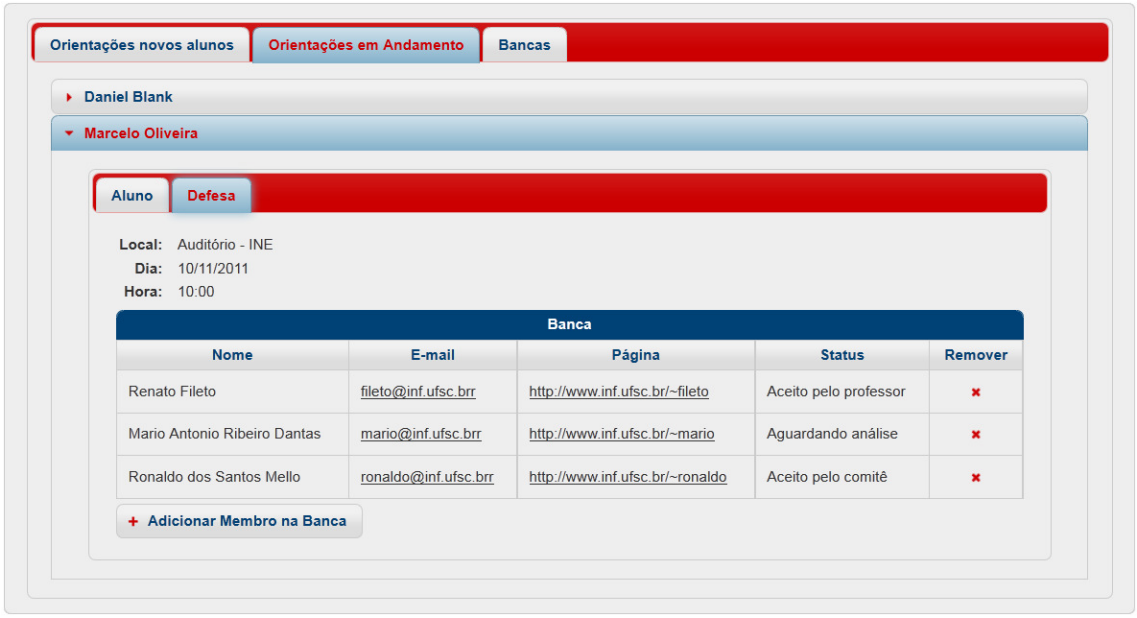

Figura 9 - Solicitação de inclusão de membro na banca.

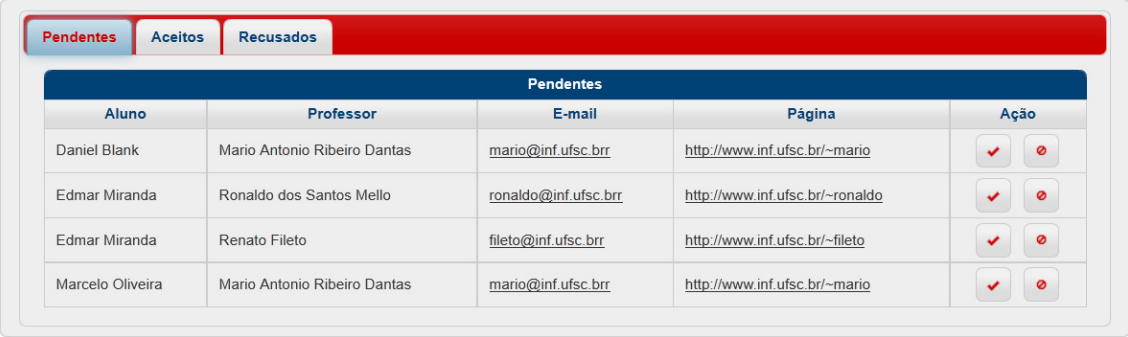

Figura 10 - Status das solicitações de participação em banca.

Figura 11 - Convites de participação em banca ao professor.

O gerenciamento das bancas possui diversos estados, que representam o processo de aceitação de um professor em uma banca. Estes estados são representados na Figura 12.

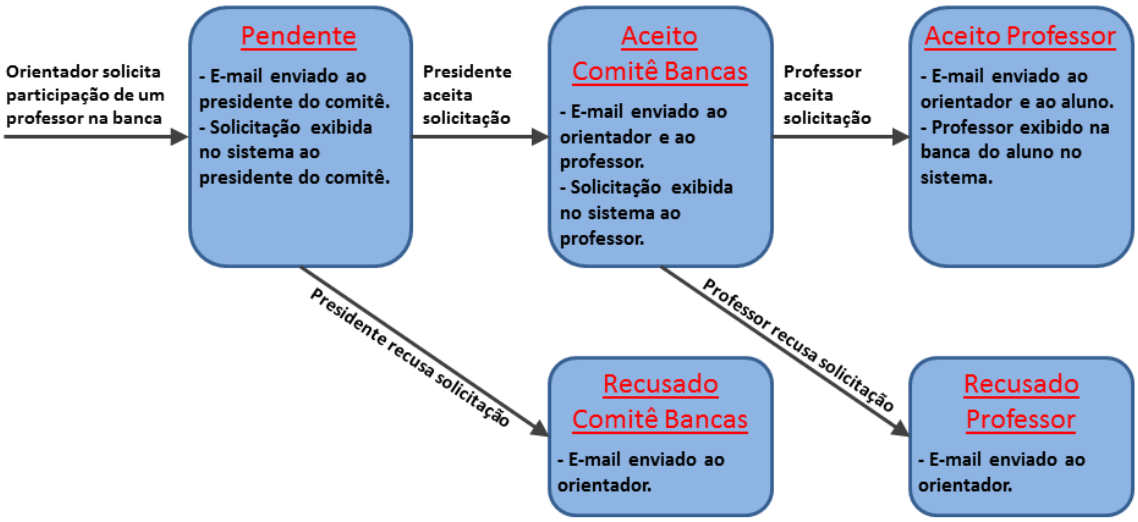

Figura 12 - Fluxo dos Estados da Banca

A seguir é descrito detalhadamente cada estado:

- **Pendente:** significa que o orientador do aluno fez uma solicitação a um professor para participar da banca de seu aluno e está aguardando a avaliação do comitê de bancas. Neste estado apenas o representante do comitê, ou seja, o presidente, é notificado via e-mail. A solicitação é exibida no sistema para o presidente.
- **Recusado pelo Comitê de Bancas:** este estado representa que o professor não foi aceito para participar da banca do aluno pelo comitê de bancas. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o comitê não aceitou o professor na banca do aluno.
- **Aceito pelo Comitê de Bancas:** este estado representa que o professor foi aceito para participar da banca do aluno pelo comitê

de bancas. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o comitê aceitou o professor na banca do aluno. O professor é notificado via e-mail que foi convidado a participar da banca. O convite é exibido no sistema ao professor.

- **Recusado pelo Professor:** significa que o professor não aceitou o convite para participar da banca do aluno. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o professor não aceitou o convite.
- **Aceito pelo Professor:** significa que o professor aceitou o convite para participar da banca do aluno. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o professor aceitou o convite. O aluno também é notificado via e-mail e no sistema o professor é adicionado em sua banca.

4.4. Demais Módulos

Os demais módulos fornecem suporte aos dois módulos apresentados anteriormente.

Gerenciador de Orientações

Este módulo tem como objetivo gerenciar o processo de orientação entre aluno e professor, auxiliando cada um dos lados e facilitando a comunicação entre eles.

Ao longo do processo de inscrição os alunos escolhem uma linha de pesquisa e um professor para orientação, sendo apenas a linha de pesquisa uma escolha obrigatória. Finalizado este processo, apenas os alunos que selecionaram a linha de pesquisa aparecem no sistema para todos os professores daquela linha. Já os que selecionaram uma linha e um professor, aparecem apenas ao professor selecionado.

A partir deste momento o professor pode selecionar alunos para orientação, ou ser for um aluno que lhe escolheu no processo de inscrição, este tem a possibilidade de liberar o aluno para a linha de pesquisa (Figura 13), ficando assim visível aos demais professores da linha para possível orientação.

Figura 13 - Orientação de alunos

Como visto na Figura 13, neste momento é possível solicitar a recomendação de alunos, módulo este que está detalhado na seção casador de perfis(seção 4.2).

Gerenciador de Entidades

Este módulo aborda o gerenciamento de todas as entidades que alimentam o sistema. Entende-se por gerenciamento, a inclusão, atualização, exclusão e busca dos dados no banco de dados. O sistema possui as seguintes entidades:

- **Aluno:** Aluno inscrito no Programa de Pós-Graduação em Ciência da Computação;
- **Professor:** Professor do Departamento de Informática e Estatística;
- **Projeto:** Projetos de pesquisa realizados pelos professores;
- **Usuário:** Um usuário do sistema, que pode estar associado a um aluno, a um professor ou não ter associação. Cada usuário possui pelo menos um nível de acesso;
- **Grupo:** Grupo associado a um professor que define quantos alunos ele poderá orientar por ano;
- **Linha de Pesquisa:** Linha de pesquisa na qual o Programa de Pós-Graduação em Ciência da Computação atua.

A Figura 14 representa o gerenciamento da entidade linha de pesquisa. Conforme visualizado na figura, é possível criar, editar e excluir uma linha de pesquisa, além de pesquisar pelas linhas através do título.

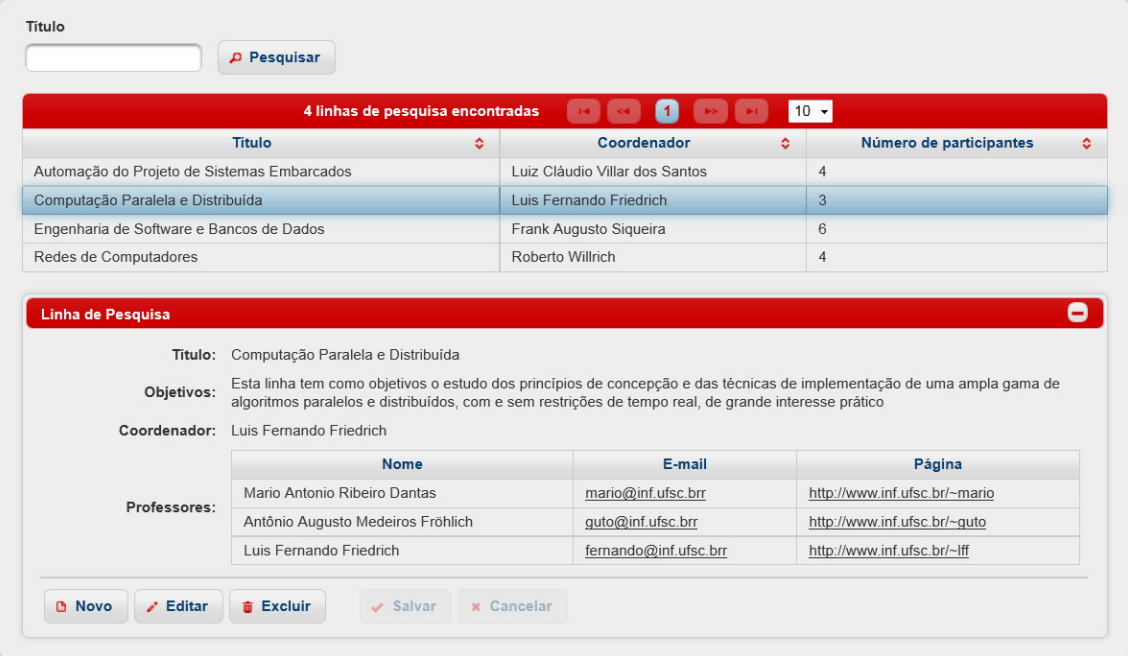

Figura 14 - Gerenciamento da entidade Linha de Pesquisa.

Módulo de Inscrição

Módulo destinado à inscrição dos alunos no sistema através do preenchimento dos seus dados divididos em diversas etapas, como, dados pessoais, dados da graduação, conhecimentos de línguas estrangeiras e currículo. Na última etapa, o usuário confere todos os seus dados informados e confirma sua inscrição. A Figura 15 mostra o processo de inscrição do aluno na etapa onde são inseridos seus dados pessoais. Todas as etapas são exibidas no topo da página para melhor acompanhamento e compreensão do que será pedido no processo de inscrição.

Na etapa Dados de Inscrição, o aluno tem a opção de solicitar recomendações de professores. Este processo está detalhado na seção casador de perfis(seção 4.2).

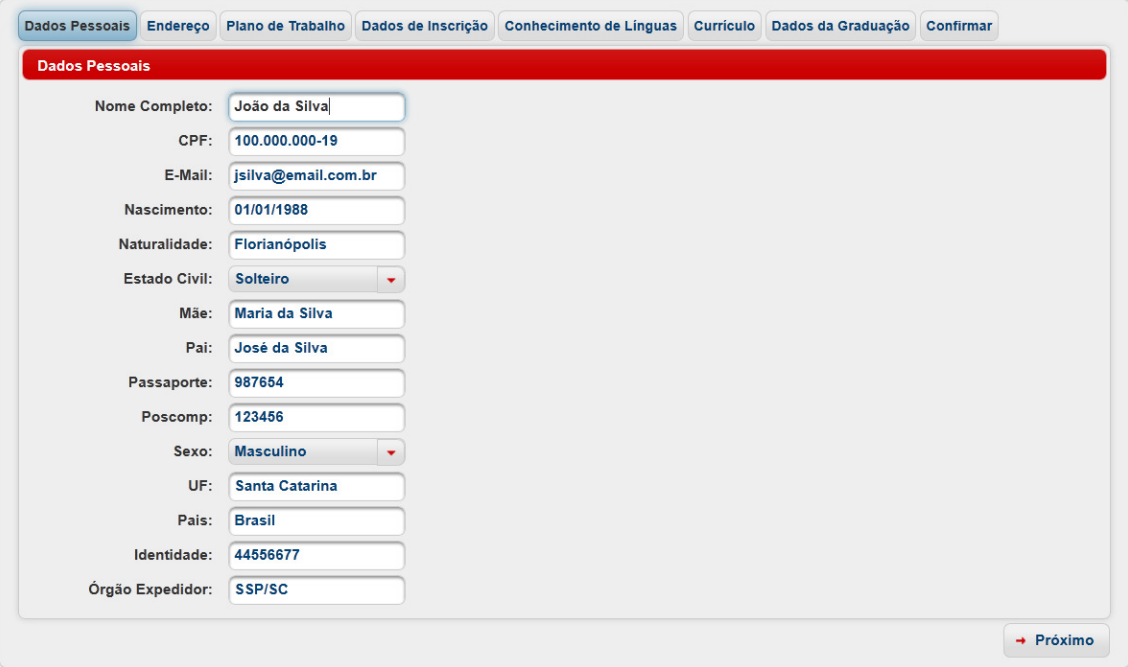

Figura 15 - Processo de inscrição do aluno.

5. Implementação

Este capítulo descreve as tecnologias utilizadas no desenvolvimento do sistema e do processo de implementação de cada módulo envolvido.

5.1. Tecnologias Utilizadas

O sistema foi desenvolvido na linguagem Java, utilizando a IDE NetBeans na plataforma Windows 7 64 bits.

O sistema segue o padrão MVC para programação Web, separando as classes em camadas de acordo com a responsabilidade de cada uma. Com o intuito de facilitar o uso deste padrão, foi utilizado o framework JavaServer Faces (JSF). O Servidor web adotado foi o Apache Tomcat 7.0.

Para simplificar a construção das interfaces foi utilizada a biblioteca de componentes JSF PrimeFaces, por possuir diversos componentes prontos para utilização.

O banco de dados escolhido para armazenar os dados do sistema foi o PostgreSQL. O framework Hibernate foi utilizado para fazer o mapeamento objeto-relacional entre os objetos da aplicação e o banco de dados. Como auxilio foi utilizado o módulo Spring ORM, que auxilia o Hibernate na gerência das sessões e transações das operações realizadas, ficando a cargo deste módulo gerenciar a abertura e o fechamento das sessões e o rollback da transação caso ocorra algum erro. Dessa forma, só é necessário se preocupar com a operação que será realizada no banco de dados.

Foram utilizados diversos outros módulos do framework Spring, como a injeção de dependência, para manter baixo o nível de acoplamento entre os módulos do sistema. Para notificar os usuários do sistema via e-mail foi utilizado o módulo Spring Mail. O módulo Spring Security foi utilizado para auxiliar e simplificar a autenticação de usuários no sistema e controlar o acesso destes às páginas restritas, através de cada nível de acesso.

5.2. Diagrama de Classes

A seguir são apresentados os diagramas de classes de dois módulos do sistema, o módulo casador de perfis (Figura 16) e o módulo gerenciador de bancas (Figura 17).

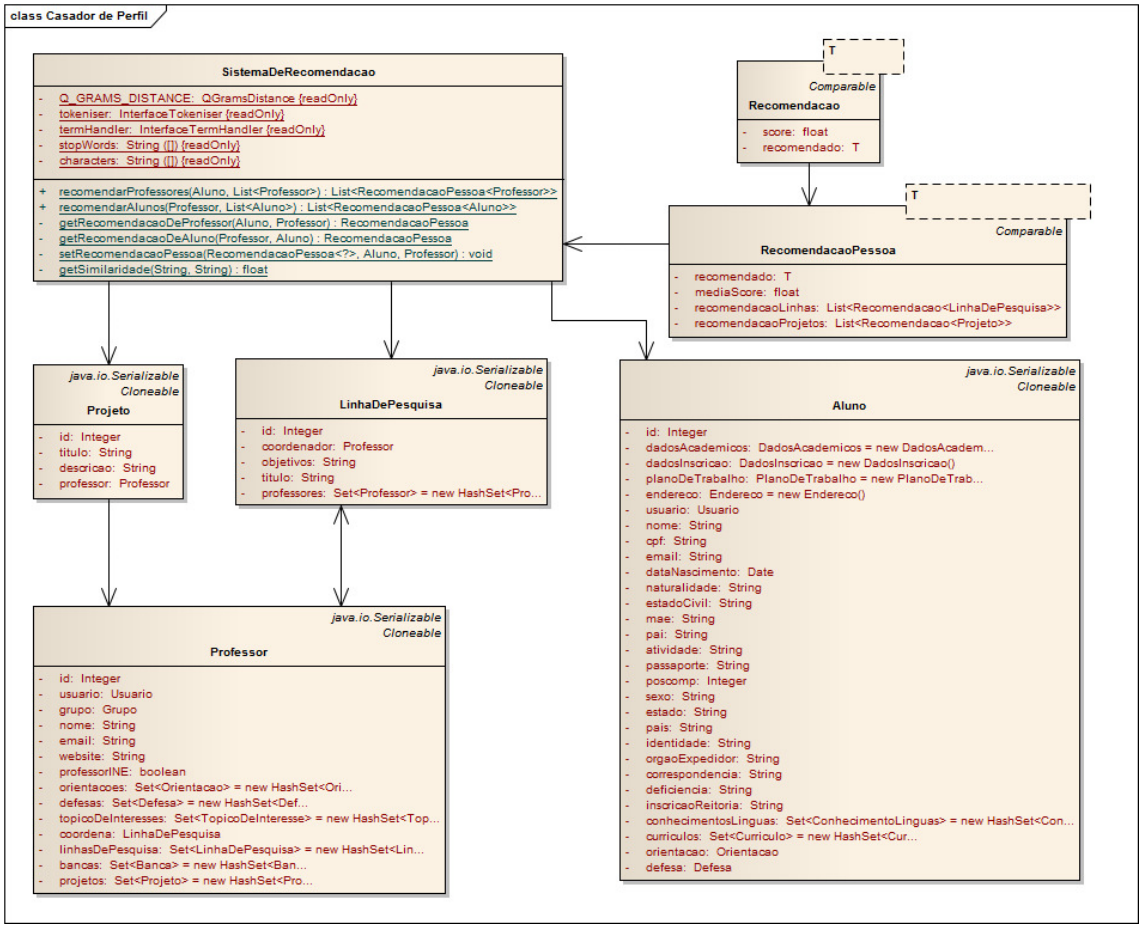

Figura 16 - Diagrama de classes do módulo casador de perfis

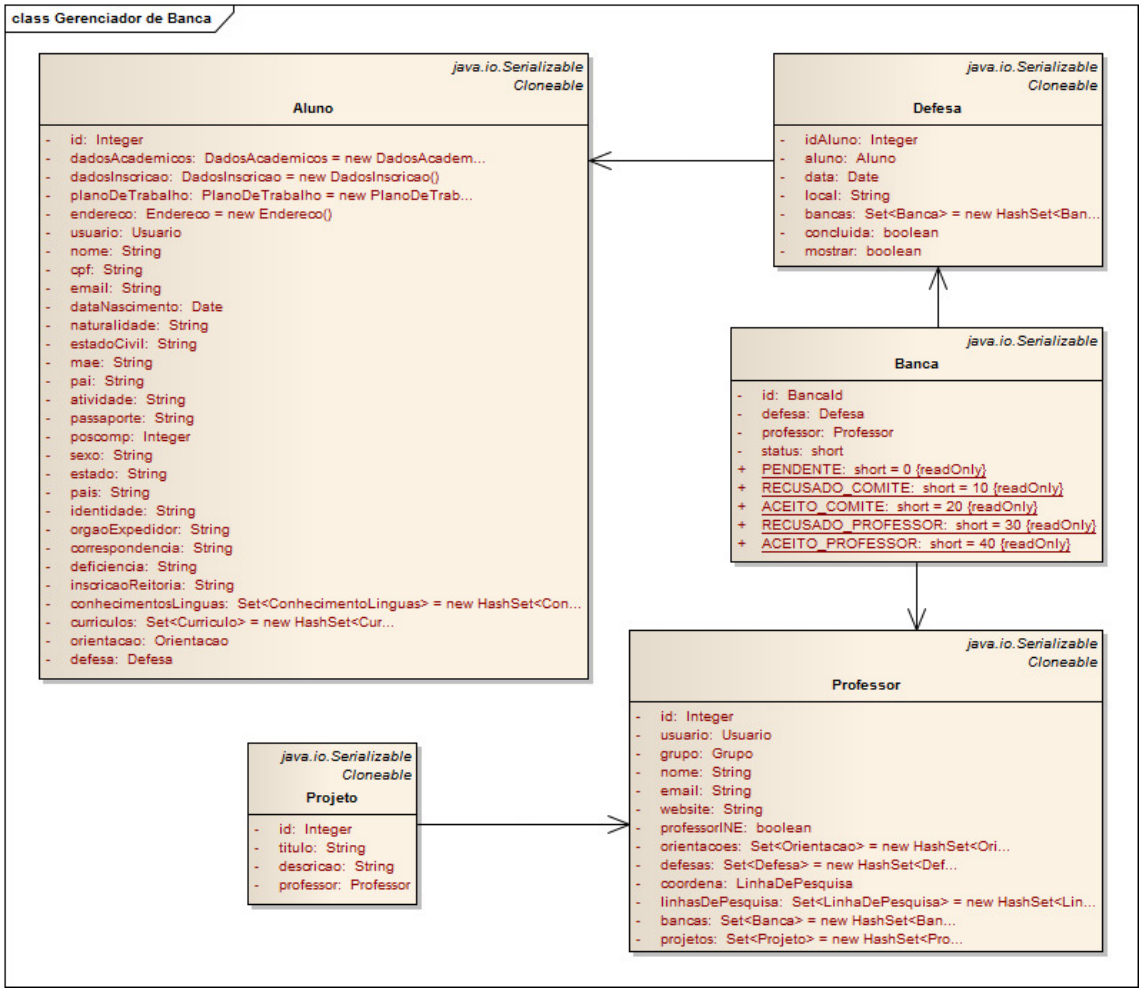

Figura 17 - Diagrama de classes do módulo gerenciador de bancas

5.3. Implementação dos Módulos

Gerenciador de Entidades

O módulo de gerenciamento das entidades do sistema está acessível apenas para os usuários com nível de acesso de administrador ou leitura, sendo que o administrador tem permissão para adicionar, editar, excluir e procurar pelos dados, e o nível de leitura apenas para pesquisar pelos dados.

As entidades do sistema são objetos denominados POJO (Plain Old Java Object), classes que contêm um construtor sem argumentos e métodos que seguem o padrão de *getters* e *setters* para seus atributos. Para cada entidade há um arquivo XML descrevendo o mapeamento entre a entidade e a tabela no banco de dados.

Cada entidade possui uma camada de negócio (Service), onde ficam as regras de negócio, uma camada de persistência (DAO – Data Access Object), que recebe uma requisição da camada de negócio e executa uma operação no banco de dados. Além destas camadas, cada entidade possui um JSF Managed Bean, conhecido também como Backing Bean, que são classes responsáveis por controlar as requisições da camada de apresentação com a camada de negócio e gerenciar os estados dos componentes das páginas.

Na camada de persistência foi utilizado o framework Hibernate para mapear as entidades para o banco de dados, facilitando assim as operações realizadas no banco. Além deste framework, foi utilizado o Spring ORM, que é um auxiliador para o Hibernate e se encarrega de gerenciar as sessões e as transações das operações.

As entidades são gerenciadas por um usuário do sistema, com exceção da inclusão do aluno. Esta operação é realizada pelo próprio aluno e sem nenhum nível de acesso, que ocorre no processo de sua inscrição.

Com o objetivo de fornecer informações sobre os professores e as linhas de pesquisa para futuros alunos do PPGCC ou interessados, as entidades Professor, Projeto e Linha de Pesquisa estão visíveis para qualquer pessoa que utilize o sistema. Já as demais entidades são restritas aos usuários com determinado nível de acesso.

O gerenciamento da entidade Usuário utiliza o Spring Mail para o envio de e-mail para novos usuários cadastrados, onde é enviado o login e a senha do usuário, sendo a senha gerada aleatoriamente e contendo oito caracteres alfanuméricos distintos.

Gerenciador de Bancas

O gerenciador de bancas possui uma entidade chamada banca, exatamente igual às descritas anteriormente. Apenas um participante do comitê, o presidente, possui um usuário no sistema com permissão para aceitar ou recusar uma solicitação de participação de professor em uma banca. Durante todo o processo de definição das bancas, as notificações recebidas via e-mail pelos usuários são realizadas utilizando o Spring Mail.

Gerenciador de Orientações

Esta entidade é criada no sistema no momento em que um professor escolhe um aluno para orientar. Após esta decisão o aluno é notificado por e-mail e o professor já está apto a gerenciar no sistema a banca do aluno.

Cada professor está relacionado a um grupo, que diz quantos alunos no máximo ele pode orientar. No caso deste número chegar ao limite e o professor tentar escolher mais um aluno para orientar, o sistema impede a orientação e avisa ao professor que o número de orientações para o seu grupo chegou ao limite.

Módulo de Inscrição

O módulo de inscrição de alunos no sistema utiliza o componente Wizard do PrimeFaces para auxiliar na coleta dos dados. Este componente cria um fluxo de trabalho através da criação de várias etapas em uma única página. À medida que a etapa atual for preenchida o usuário poderá ir para a etapa seguinte, ou voltar para a etapa anterior caso queira alterar algum dado. A etapa seguinte somente será exibida se a etapa atual passar pelo processo de validação. Por exemplo, se um dado for definido com sendo obrigatório e não for preenchido pelo usuário, o sistema irá exibir uma mensagem e permanecerá na mesma etapa. Caso contrário, irá para a etapa seguinte. Na última etapa são exibidos todos os dados informados pelo usuário, onde ele poderá retornar para etapas anteriores para alterar algum dado ou confirmar a sua inscrição no sistema. Ao final do processo, antes da confirmação dos dados, é solicitada a inserção de palavras para verificar que o usuário é humano, evitando que softwares automatizados executem a inscrição de usuários.

Módulo de Segurança

Módulo responsável pela autenticação de um usuário no sistema e do gerenciamento das páginas acessíveis de acordo com o nível de acesso do usuário. Ele não permite que um usuário acesse páginas que não tem autorização, redirecionando-o para a página de login do sistema. Para

aumentar a segurança, foi utilizado o algoritmo de hash MD5 para cifrar as senhas de usuários para trafegarem com segurança pela rede.

O módulo de segurança aborda o processo de login de um usuário no sistema e o controle dos níveis de acesso que são necessários para um usuário poder visualizar certa página. Para facilitar estes processos foi utilizado o Spring Security, que utiliza um arquivo XML para sua configuração. No XML vão informações das páginas e os níveis de acessos necessários para visualizá-las, a página que é realizada o login, a página que o usuário será redirecionado caso ocorra falha na autenticação ou se a autenticação foi realizada com sucesso ou a página após fazer logout, além de consultas ao banco de dados para buscar o login e a senha do usuário e os seus respectivos níveis de acessos para verificar a autenticidade dos dados informados pelo usuário. Caso a autenticação foi realizada com sucesso, o Spring Security armazena as informações do usuário na sessão, como o seu login e seus níveis de acesso. Antes de acessar uma página ele certifica se há a necessidade de possuir um determinado nível de acesso. Caso exista, ele irá verificar se há um usuário na sessão e se este possui o nível de acesso requerido.

O sistema possui cinco níveis de acesso, que são:

- **Administrador:** usuário poderá acessar todas as áreas do sistema. Ele terá permissão para incluir, alterar, excluir e visualizar todas as entidades do sistema.
- **Leitura:** usuário poderá acessar todas as áreas do sistema. Terá apenas permissão para visualizar as entidades do sistema.

- **Professor:** usuário poderá visualizar e alterar os seus dados e gerenciar suas orientações de mestrado.
- **Aluno:** usuário poderá visualizar e alterar os seus dados e visualizar os dados de sua defesa de mestrado.
- **Presidente Banca:** usuário tem acesso à gerência das bancas, podendo aceitar ou recusar professores para participação em bancas.

6. Conclusões e Trabalhos Futuros

O desenvolvimento deste trabalho trouxe como aprendizado o funcionamento dos processos dentro do Programa de Pós-Graduação em Ciência da Computação, tais como o processo de inscrição de um aluno, o processo de composição das bancas de mestrado e o processo de orientação de alunos. Além disso, foram abordados sistemas de recomendação, que proporcionaram um grande aprendizado neste assunto, como os seus objetivos, as diversas formas de realizar as recomendações e coletar as informações, bem como a análise de alguns sistemas de recomendações disponíveis na internet.

O sistema desenvolvido procura centralizar as informações e os processos envolvidos durante o período de mestrado dos alunos, fazendo com que as tarefas realizadas se tornem mais fáceis de serem controladas e gerenciadas, além de serem mais ágeis e menos burocráticas, devido à ausência do uso de papéis. Através do sistema de recomendação implantado dentro da aplicação os alunos recebem um grande auxílio na decisão de um orientador, já que existem muitos professores no departamento do INE e cada um tem diversos projetos e participa de uma ou mais linhas de pesquisa. Os professores também se beneficiam com este sistema, pois recebem recomendações de alunos que mais se assemelham às suas linhas de pesquisa e projetos.

Os objetivos propostos por este trabalho foram todos concretizados, porém podemos sugerir novas funcionalidades ao sistema para torná-lo mais completo, sendo elas:

- Um sistema de Log para registrar as ações realizadas pelos usuários no sistema, como por exemplo, inclusão, alteração e exclusão de dados, além de registrar eventuais erros ocorridos no sistema;
- Melhorar a interação do aluno com o professor permitindo o compartilhamento de arquivos para melhor acompanhamento dos envolvidos no processo de defesa de dissertação;
- Inclusão de funcionalidades para a realização de backup do banco de dados via interface do sistema, além da possibilidade de poder restaurar o banco de dados em caso de perda ou comprometimento do mesmo.

7. Referências Bibliográficas

DEMONSTRAÇÃO DOS COMPONENTES DO PRIMEFACES. Disponível em: <http://www.primefaces.org/showcase-labs/ui/home.jsf>. Acesso em: 26 de ago. 2011.

DOCUMENTAÇÃO HIBERNATE. Disponível em: <http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/>. Acesso em: 26 ago. 2011.

DOCUMENTAÇÃO POSTGRESQL. Disponível em: <http://www.postgresql.org/docs/9.0/interactive/index.html>. Acesso em: 26 de ago. 2011.

eHarmony. Disponível em: < http://www.eharmony.com.br/>. Acesso em: 2 de jul. 2011.

GONÇALVES, Edson. Segurança com Spring Security 3.0 utilizando banco de dados. Disponível em: <http://imasters.com.br/artigo/16780/seguranca/seguranca_com_spring_security_30_utilizando_banco_de_dados/>. Acesso em: 26 de ago. 2011.

JAVADOC JAVASERVER FACES. Disponível em: <http://javaserverfaces.java.net/nonav/docs/2.0/javadocs/>. Acesso em: 26 de ago. 2011.

JAVADOC HIBERNATE. Disponível em: <http://docs.jboss.org/hibernate/core/3.6/javadocs/>. Acesso em: 26 de ago. 2011.

JAVADOC PRIMEFACES. Disponível em: <http://primefaces.prime.com.tr/docs/tag/>. Acesso em: 26 de ago. 2011.

JAVADOC SPRING. Disponível em: <http://static.springsource.org/spring/docs/3.1.0.M1/javadoc-api/>. Acesso em: 26 ago. 2011.

JAVADOC SPRING SECURITY. Disponível em: <http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/index.html>. Acesso em: 26 ago. 2011.

JAVA WEB DEVELOPMENT TUTORIALS. Disponível em: <http://www.mkyong.com/>. Acesso em: 26 de ago. 2011.

MovieLens. Disponível em: < http://www.movielens.org/>. Acesso em: 5 jul. 2011.

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO. Disponível em: <http://www.ppgcc.inf.ufsc.br/>. Acesso em: 25 nov. 2010.

REATEGUI, E.B.; CAZELLA, S.C. Sistemas de Recomendação. Congresso da Sociedade Brasileira de Computação – CSBC. São Leopoldo/RS. 2005.

SISTEMA DE COORDENAÇÃO DE PROJETOS DO INE. Disponível em: <http://projetos.inf.ufsc.br/>. Acesso em: 26 ago. 2011.

SISTEMA DE RECOMENDAÇÃO. Disponível em: <http://pt.wikipedia.org/wiki/Sistema_de_recomenda%C3%A7%C3%A3o>. Acesso em: Acesso em: 26 ago. 2011.

8. Anexo A – Artigo

SWAP
Um sistema web para inscrição e auxílio às atividades do Programa de Pós-Graduação em Ciência da Computação

Daniel Blank

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brasil

dblank@inf.ufsc.br

Abstract. This study aims to develop a web application to help students in the inscription in the Graduate Program in Computer Science, as well as assist students and teachers in the program activities. Through a recommendation system, the application is able to recommend which teachers are best suited to the student at the time of your inscription , depending on the areas of interest of each. After the registration of students, teachers may also request recommendations, assisting in the selection of students to guide. Other features that the application is intended to is assist the processes of composition of boards of master and the imposition of limits of students that a teacher may guide. In order to facilitate and expedite the work of those responsible for these processes, eliminating bureaucratic parties, successive exchanges of emails and organization of information in printed documents, these processes were fully automated.

Resumo. Este trabalho tem como objetivo o desenvolvimento de uma aplicação web para auxiliar os alunos na inscrição ao mestrado no Programa de Pós-Graduação em Ciência da Computação (PPGCC), bem com auxiliar alunos e professores nas atividades do programa. Através de um sistema de recomendação, a aplicação é capaz de recomendar quais os professores são mais indicados para o aluno na hora da sua inscrição, dependendo das áreas de interesses de cada um. Após as inscrições dos alunos, os professores também poderão solicitar recomendações, auxiliando na escolha de alunos para orientar. Outras funcionalidades que a aplicação visa auxiliar são os processos de composição das bancas de mestrado e a imposição dos limites de alunos que um professor pode orientar. Com o intuito de facilitar e agilizar o trabalho dos responsáveis nestes processos, acabando com as partes burocráticas, trocas sucessivas de e-mails e organização das informações em documentos impressos, estes processos foram totalmente automatizados.

1. Introdução

Para inscrever-se no mestrado do PPGCC, o candidato deve ser portador de diploma de nível superior compatível com a área de interesse, além de ter que realizar o "Exame Nacional para Ingresso na Pós-Graduação em Computação" (POSCOMP). Para

realizar a inscrição no processo seletivo do Programa, os candidatos devem preencher a Ficha de Inscrição disponível no site do Sistema de Controle Acadêmico da Pós-Graduação (http://www.capg.ufsc.br/inscricao). Durante a inscrição no sistema o aluno deve escolher um professor para orientá-lo durante o período do mestrado, porém, se não for escolhido nenhum professor, cabe a algum professor escolhê-lo após o processo de inscrição. As bancas de mestrado só podem ser formadas por professores doutores. Diferentemente do processo de mestrado, o ingresso na graduação requer o ensino médio completo e a realização da prova do vestibular. Após isso, se o aluno for aprovado, é necessário apenas realizar a matrícula na universidade.

O processo de seleção de novos alunos para ingresso no PPGCC, em nível de mestrado, é baseado no número de vagas disponível para cada professor de cada linha de pesquisa. O número de vagas é determinado de acordo com uma classificação previamente efetuada pela Comissão de Produção Científica - CPC.

Para o aluno descobrir qual o professor mais adequado ao seu plano de trabalho no mestrado, é necessário que ele pesquise por este professor no site do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da UFSC . O caso do professor escolher alunos é mais complicado, pois é necessário que ele vá à secretaria do curso e verifique nos documentos de inscrição impressos de cada aluno se algum se encaixa na sua linha de pesquisa. Ambos os processos implicam em tempo disponível e paciência, principalmente no caso do professor. Para incluir um membro na banca de mestrado, é necessário comunicar ao coordenador do programa para junto com uma comissão avaliar se o membro está apto ou não a fazer parte da banca. Este processo é burocrático e pode demandar algum tempo para ser concretizado. Os professores têm um determinado limite de orientações, o qual é controlado através de papéis, tornando o controle mais difícil e suscetível a falhas por parte do responsável.

O preenchimento das inscrições dos candidatos não é feita em um site administrado pelo INE. Os dados requeridos para a inscrição são restritos comparado ao ideal para o INE, além de dificultar o acesso e a manipulação dos dados dos candidatos. No sistema atual não é possível visualizar a nota do aluno no POSCOMP e nem identificar se um aluno entrou em contato com algum professor previamente.

Todo o processo de definição de qual professor orientará quais alunos e o limite de alunos por professor, além do processo de elaboração das bancas de mestrado e aceitação dos professores nelas, são realizados em papel ou por trocas de e-mails entre os envolvidos. Estes processos acarretam em diversos problemas, como a dificuldade do responsável em organizar e administrar quais alunos já tem orientador definido, quantos alunos um professor ainda poderá orientar, a composição formalizada de cada banca, etc.

Devido a grande dificuldade das pessoas em tomar decisões quanto à escolha de uma grande diversidade de itens e recursos ou da falta de conhecimento das opções, foram propostos os sistemas de recomendações. O objetivo destes sistemas é auxiliar o usuário na escolha do que ele deseja. Sem este auxílio o usuário poderia levar muito tempo até encontrar o que procura, ou até não encontrar. Os sistemas de recomendações visam filtrar informações que possam ser de interesse do usuário com base nas suas informações e características, e a partir delas recomendar-lhe itens e recursos.

Objetivos

Desta forma, propõe-se o desenvolvimento de um sistema de recomendação com o objetivo de auxiliar os alunos na escolha de um professor para orientação, recomendando professores que mais se assemelham ao plano de trabalho do aluno, sem que este precise analisar professor por professor até encontrar o mais adequado. Para os professores, o sistema auxilia recomendando alunos que mais combinam com suas linhas de pesquisa e projetos, evitando assim que o professor necessite procurar exaustivamente por alunos em documentos impressos. Outra situação que o sistema visa eliminar é o processo trabalhoso do responsável na gestão das bancas. Para evitar o uso de papéis e múltiplas trocas de e-mails, o sistema propõe um processo automatizado para definir as bancas de uma forma mais rápida e eficiente para os envolvidos neste processo.

Os objetivos específicos deste trabalho são:

- Auxiliar o candidato na inscrição no Programa de Pós-Graduação em Ciência da Computação (PPGCC);
- Auxiliar o aluno na escolha de um orientador, através de recomendação;
- Auxiliar o professor na escolha de um aluno para orientar, através de recomendação;
- Simplificar o processo de definição das bancas de mestrado.

Motivação

A principal motivação para a realização deste trabalho teve como intuito auxiliar no processo de inscrição do aluno, pois o mesmo pode não saber qual área seguir ou qual professor se adequa mais com os seus interesses. Neste caso, o aluno pode levar um tempo até analisar as linhas de pesquisa e os professores para tomar sua decisão. Após o processo de inscrição, os professores podem analisar os alunos que ainda não contataram algum professor, o que pode levar certo tempo também, já que é preciso analisar os interesses de cada um.

O processo de composição das bancas de mestrado demanda muito trabalho ao responsável, pois ele precisa lidar com diversas solicitações de inclusão de membros em bancas, tendo que realizar sucessivas trocas de e-mails. O processo da imposição dos limites de alunos que um professor pode orientar é realizado com documentos impressos, tornando o controle e organização dos documentos complicado, podendo perdê-los ou levar muito tempo até encontrar qual está procurando.

Este trabalho é organizado da seguinte forma. No Capítulo 2 é apresentada uma visão geral dos sistemas de recomendações, descrevendo as formas de realizar recomendações, os tipos de coleta de informações e as formas de disponibilização dos itens. O Capítulo 3 apresenta aplicações que utilizam sistemas de recomendações e o sistema web para suporte à coordenação de projetos do INE. A apresentação do sistema proposto é realizada no Capítulo 4, mostrando um mapa arquitetural do funcionamento

do sistema e a descrição geral dividida em diversos módulos. No Capítulo 5 é abordada a forma de implementação do sistema, mostrando as tecnologias utilizadas para o desenvolvimento, além da descrição técnica do desenvolvimento de cada módulo. Finalmente, no Capítulo 6 são descritas as conclusões referentes a este trabalho, descrevendo os aprendizados que este trouxe, além das sugestões para trabalhos futuros a serem desenvolvidos como complemento para este trabalho.

2. Sistemas de Recomendação

Sistemas de Recomendação são classificados em três grupos de acordo com a forma na qual ele realiza as recomendações. O primeiro, denominado de Recomendações Baseadas em Conteúdo, baseia-se na recomendação de itens semelhantes aos já desejados pelo usuário, ou seja, analisa as semelhanças entre seus itens desejados com os demais itens disponíveis. Os interesses dos usuários são diferenciados, recomendando itens individualmente para cada usuário. O segundo grupo, denominado de Recomendações Colaborativas, baseia-se na recomendação de itens que foram desejados por usuários com interesses semelhantes ao usuário que realiza a consulta. Os interesses dos usuários são agrupados, criando comunidades de usuários que possuem interesses semelhantes. Este grupo se difere do primeiro por não exigir o reconhecimento do conteúdo dos itens. O último grupo de sistemas de recomendações, conhecido como Métodos Híbridos, aborda a combinação dos dois grupos citados anteriormente, com o objetivo de construir um sistema mais eficiente.

A coleta das informações referentes a um usuário pode ser realizada de forma explícita ou implícita. Na forma explícita, o usuário especifica seus interesses através do preenchimento de algum formulário. Já na forma implícita, seus interesses são coletados sem sua percepção, pois a coleta é realizada com base nos itens visitados e desejados por ele.

As recomendações de itens e usuários são disponibilizadas de acordo com a forma que são apresentadas no sistema, sendo dividido em três formas. A primeira denominada Push, consiste em recomendar itens ao usuário sem que ele esteja interagindo com o sistema, como por exemplo, enviar e-mail ao usuário com recomendações de ofertas. A segunda, conhecida como Pull ocorre apenas quando o usuário solicita tal recomendação, possibilitando o controle de quando serão exibidas. A última forma de apresentação é denominada Passiva, onde as recomendações são exibidas no contexto do sistema, como por exemplo, itens mais vendidos de uma loja.

A seguir será apresentado como os sistemas de recomendação são classificados. Primeiramente será abordado as diferentes formas de como são realizadas as recomendações, em seguida, as formas de coletar informações do usuário, e finalmente as diferentes maneiras de disponibilizar as recomendações ao usuário.

3. SWAP

Este capítulo aborda a proposta de um sistema de recomendação que possui dois focos principais: (i) auxiliar alunos e professores no processo de inscrição no PPGCC através de um sistema de recomendação; e (ii) gerir as bancas de mestrado de uma forma prática e simplificar os processos de comunicação entre as partes envolvidas.

A forma de recomendação utilizada no desenvolvimento do sistema é a recomendação baseada em conteúdo pelo motivo de ser a mais adequada ao sistema proposto, pois como é uma relação direta entre aluno e professor, não se faz necessário criar grupos de usuários afins, mas sim comparar diretamente os conteúdos de alunos com os de professores e vice-versa.

O sistema utiliza a coleta explícita de informação para realizar as recomendações, pois como são utilizadas informações pessoais, o sistema não conseguiria extrair estas informações. Logo é necessário que o usuário insira explicitamente as informações no sistema.

A disponibilidade das recomendações é feita utilizando a técnica pull, para assim fornecer ao usuário o controle de quando as recomendações são realizadas.

Arquitetura

A Figura 1 apresenta o funcionamento de toda a aplicação, demonstrando esquematicamente os fluxos dos processos e a integração entre os módulos presentes no sistema. Nesta figura é possível ver os módulos presentes na aplicação e a interação entre eles. O módulo de segurança, responsável pela autenticação do usuário no sistema, se comunica diretamente com o banco de dados para conferir se os dados envidados pelo usuário são válidos. O gerenciador de entidades é responsável por realizar as operações das entidades com o banco de dados, como salvar, atualizar, excluir e pesquisar por uma entidade. O módulo de inscrição é encarregado pelo processo de inscrição do aluno, sendo que no final deste processo o aluno é inserido no banco de dados. Ainda neste módulo é possível solicitar recomendações de professores pelo módulo casador de perfis. Outro módulo que utiliza o casador de perfis é o gerenciador de orientações, neste caso recomendando alunos ao professor que solicita pelo serviço.

O gerenciador de orientações lida com o processo de escolha de alunos para orientação pelos professores e com o limite de orientações imposto pelo grupo de cada professor. O gerenciador de bancas é responsável por todo o processo de composição de uma banca de mestrado. O processo consiste na solicitação de inclusão de um membro na banca, seguido pela análise de um comitê de bancas e por fim a análise do professor convidado. Os módulos mais importantes deste sistema são o gerenciador de bancas e o casador de perfis.

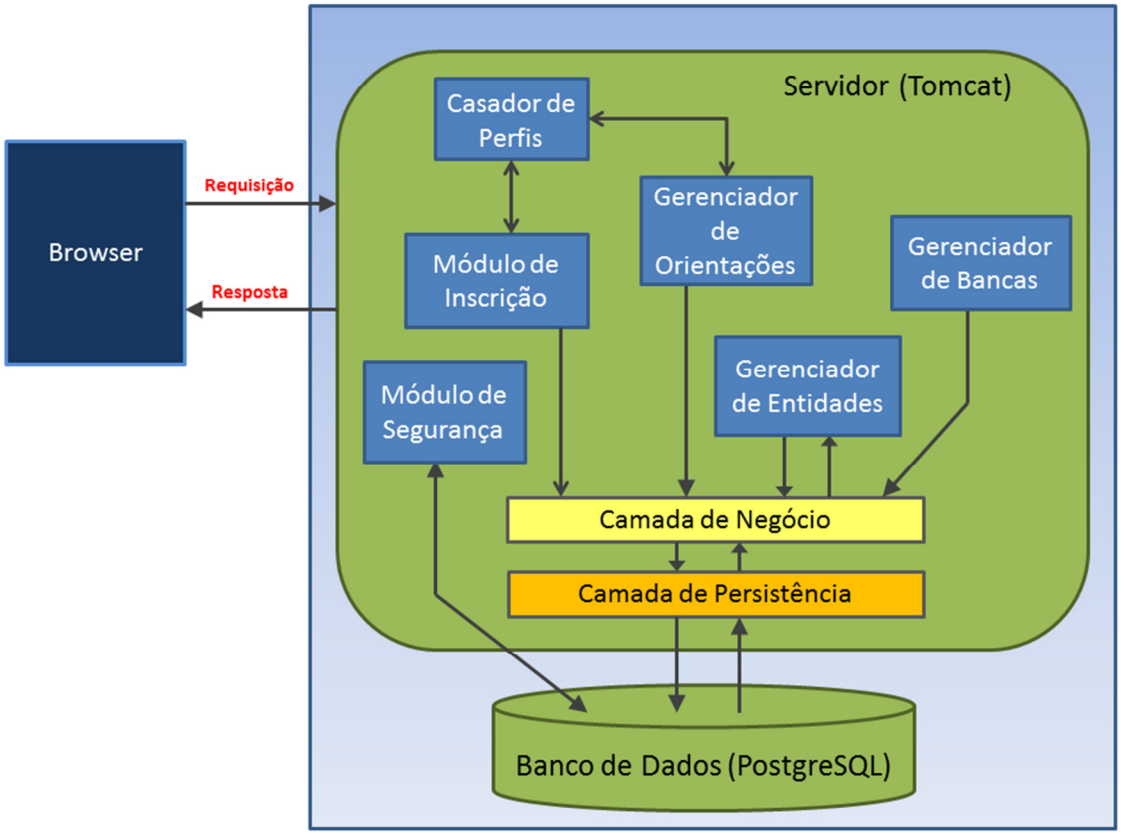

Figura 1 - Mapa Arquitetural da Aplicação

Casador de Perfis

Módulo responsável por auxiliar alunos na escolha do orientador com mais afinidade ao seu plano de trabalho e para professores na escolha de alunos que se encaixam com os seus projetos e linha de pesquisa.

Para os alunos, este módulo está presente durante o seu processo de inscrição. O aluno informa o seu plano de trabalho para o mestrado e na etapa seguinte do processo de sua inscrição, (onde ele seleciona a linha de pesquisa e o professor para possível orientação), há a opção de solicitar recomendações de professores. O sistema de recomendação se baseia na descrição do plano de trabalho informado pelo aluno com os objetivos das linhas de pesquisa e com as descrições dos projetos de cada professor. Considerando esta análise o sistema lista os professores melhores classificados. A classificação é realizada através da média de scores do plano de trabalho com as linhas de pesquisas e projetos utilizando a função de similaridade Q-Grams Distance, que gera um score de zero até um, sendo que quanto mais próximo de um, mais similares são os textos comparados. A Figura 2 apresenta a página no sistema onde são mostrados os professores recomendados, onde o aluno tem acesso às linhas de pesquisa e projetos de cada professor recomendado, auxiliando assim o aluno na escolha de um orientador.

Figura 2 - Recomendação de professores no processo de inscrição do aluno.

Para os professores, este módulo está presente durante o processo de escolha de alunos para orientação. O professor solicita recomendações de alunos e com base nas linhas de pesquisa e dos projetos do professor, o sistema analisa a similaridade com o plano de trabalho dos alunos que durante o processo de inscrição selecionaram uma linha de pesquisa do professor e/ou o próprio professor. A partir desta análise, o sistema classifica os alunos com base na média do score obtido na análise, usando a mesma função de similaridade e processo de análise descritos anteriormente na recomendação de professores a alunos. A Figura 3 mostra a área no sistema onde o professor seleciona alunos para orientação. Há a opção de solicitar a recomendação de alunos que mais se assemelham com suas linhas de pesquisa e projetos.

Figura 3 - Recomendação de alunos no processo de escolha de orientandos.

Gerenciador de Bancas

Este módulo é responsável pela gestão das bancas de defesa de mestrado, onde cabe a um comitê avaliar se um professor está apto a participar de uma defesa.

Os orientadores efetuam no sistema uma solicitação para inclusão de cada membro em uma banca (Figura 4). O presidente do Comitê de bancas, designado a avaliar as solicitações, é notificado e em seguida avalia junto ao comitê se o professor está apto ou não a participar da banca (Figura 5). Caso o professor seja aceito na banca, este é notificado sobre a sua participação e decide se possui interesse em participar ou não (Figura 6). O orientador é notificado de qualquer decisão ocorrida neste processo, ou seja, se o professor foi ou não aceito pelo comitê e se o professor aceitou ou não participar da banca.

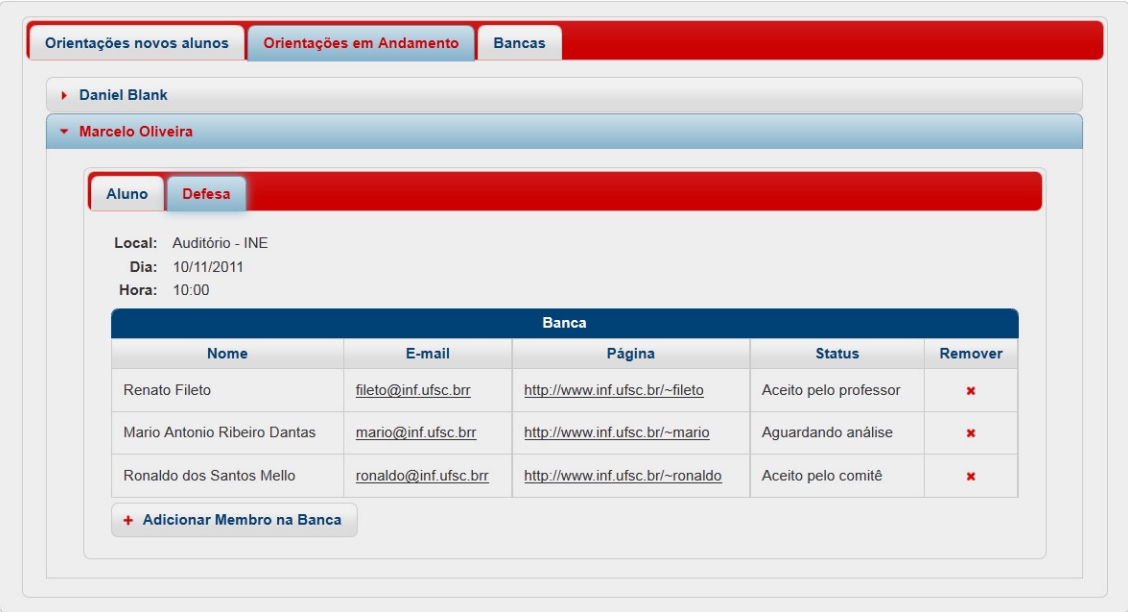

Figura 4 - Solicitação de inclusão de membro na banca.

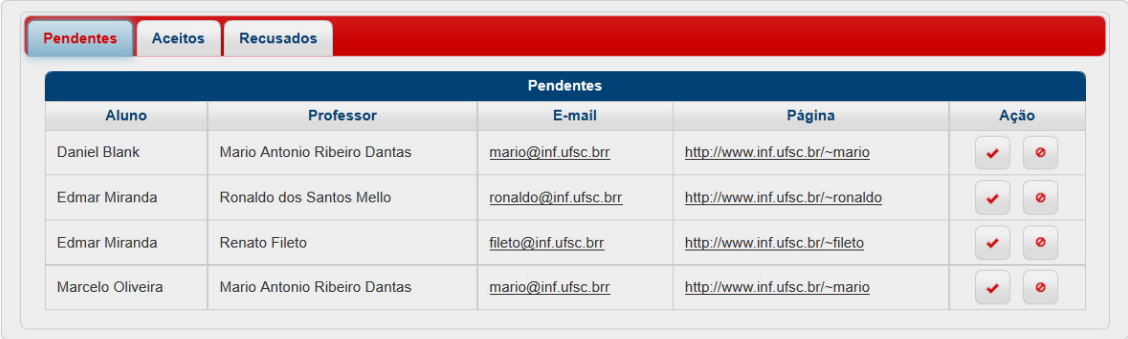

Figura 5 - Status das solicitações de participação em banca.

Figura 6 - Convites de participação em banca ao professor.

O gerenciamento das bancas possui diversos estados, que representam o processo de aceitação de um professor em uma banca. Estes estados são representados na Figura 7.

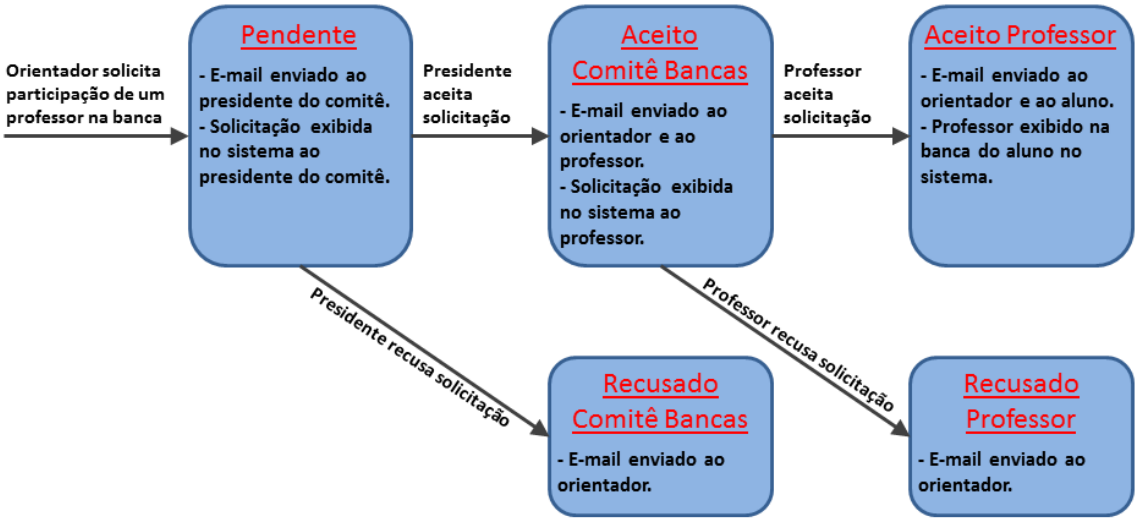

Figura 7 - Fluxo dos Estados da Banca

A seguir é descrito detalhadamente cada estado:

- **Pendente:** significa que o orientador do aluno fez uma solicitação a um professor para participar da banca de seu aluno e está aguardando a avaliação do comitê de bancas. Neste estado apenas o representante do comitê, ou seja, o presidente, é notificado via e-mail. A solicitação é exibida no sistema para o presidente.

- **Recusado pelo Comitê de Bancas:** este estado representa que o professor não foi aceito para participar da banca do aluno pelo comitê de bancas. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o comitê não aceitou o professor na banca do aluno.

- **Aceito pelo Comitê de Bancas:** este estado representa que o professor foi aceito para participar da banca do aluno pelo comitê de bancas. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o comitê aceitou o professor na banca do aluno. O professor é notificado via e-mail que foi convidado a participar da banca. O convite é exibido no sistema ao professor.

- **Recusado pelo Professor:** significa que o professor não aceitou o convite para participar da banca do aluno. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o professor não aceitou o convite.

- **Aceito pelo Professor:** significa que o professor aceitou o convite para participar da banca do aluno. O orientador é notificado via e-mail desta decisão e atualizado no sistema o estado da solicitação de participação na banca, dizendo que o professor aceitou o convite. O aluno também é notificado via e-mail e no sistema o professor é adicionado em sua banca.

Demais Módulos

Os demais módulos fornecem suporte aos dois módulos apresentados anteriormente.

Gerenciador de Orientações

Este módulo tem como objetivo gerenciar o processo de orientação entre aluno e professor, auxiliando cada um dos lados e facilitando a comunicação entre eles.

Ao longo do processo de inscrição os alunos escolhem uma linha de pesquisa e um professor para orientação, sendo apenas a linha de pesquisa uma escolha obrigatória. Finalizado este processo, apenas os alunos que selecionaram a linha de pesquisa aparecem no sistema para todos os professores daquela linha. Já os que selecionaram uma linha e um professor, aparecem apenas ao professor selecionado.

A partir deste momento o professor pode selecionar alunos para orientação, ou ser for um aluno que lhe escolheu no processo de inscrição, este tem a possibilidade de liberar o aluno para a linha de pesquisa (Figura 8), ficando assim visível aos demais professores da linha para possível orientação.

Figura 8 - Orientação de alunos

Como visto na Figura 8, neste momento é possível solicitar a recomendação de alunos, módulo este que está detalhado na seção casador de perfis.

Gerenciador de Entidades

Este módulo aborda o gerenciamento de todas as entidades que alimentam o sistema. Entende-se por gerenciamento, a inclusão, atualização, exclusão e busca dos dados no banco de dados. O sistema possui as seguintes entidades:

- **Aluno:** Aluno inscrito no Programa de Pós-Graduação em Ciência da Computação;
- **Professor:** Professor do Departamento de Informática e Estatística;
- **Projeto:** Projetos de pesquisa realizados pelos professores;

- **Usuário:** Um usuário do sistema, que pode estar associado a um aluno, a um professor ou não ter associação. Cada usuário possui pelo menos um nível de acesso;
- **Grupo:** Grupo associado a um professor que define quantos alunos ele poderá orientar por ano;
- **Linha de Pesquisa:** Linha de pesquisa na qual o Programa de Pós-Graduação em Ciência da Computação atua.

A Figura 9 representa o gerenciamento da entidade linha de pesquisa. Conforme visualizado na figura, é possível criar, editar e excluir uma linha de pesquisa, além de pesquisar pelas linhas através do título.

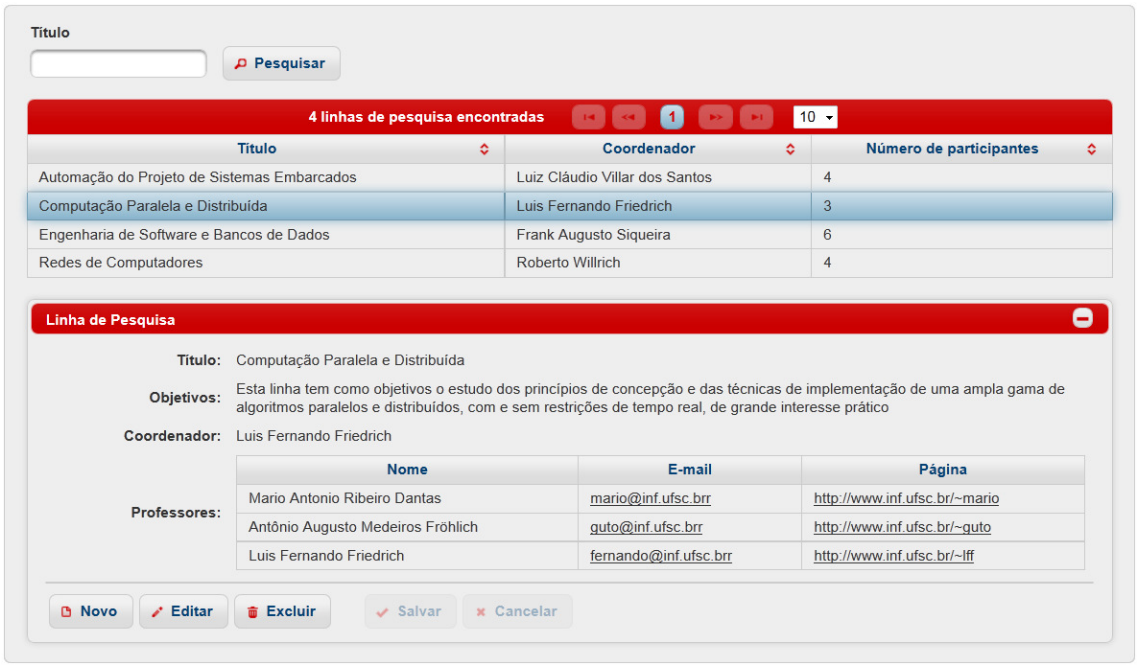

Figura 9 - Gerenciamento da entidade Linha de Pesquisa.

Módulo de Inscrição

Módulo destinado à inscrição dos alunos no sistema através do preenchimento dos seus dados divididos em diversas etapas, como, dados pessoais, dados da graduação, conhecimentos de línguas estrangeiras e currículo. Na última etapa, o usuário confere todos os seus dados informados e confirma sua inscrição. A Figura 10 mostra o processo de inscrição do aluno na etapa onde são inseridos seus dados pessoais. Todas as etapas são exibidas no topo da página para melhor acompanhamento e compreensão do que será pedido no processo de inscrição.

Na etapa Dados de Inscrição, o aluno tem a opção de solicitar recomendações de professores. Este processo está detalhado na seção casador de perfis.

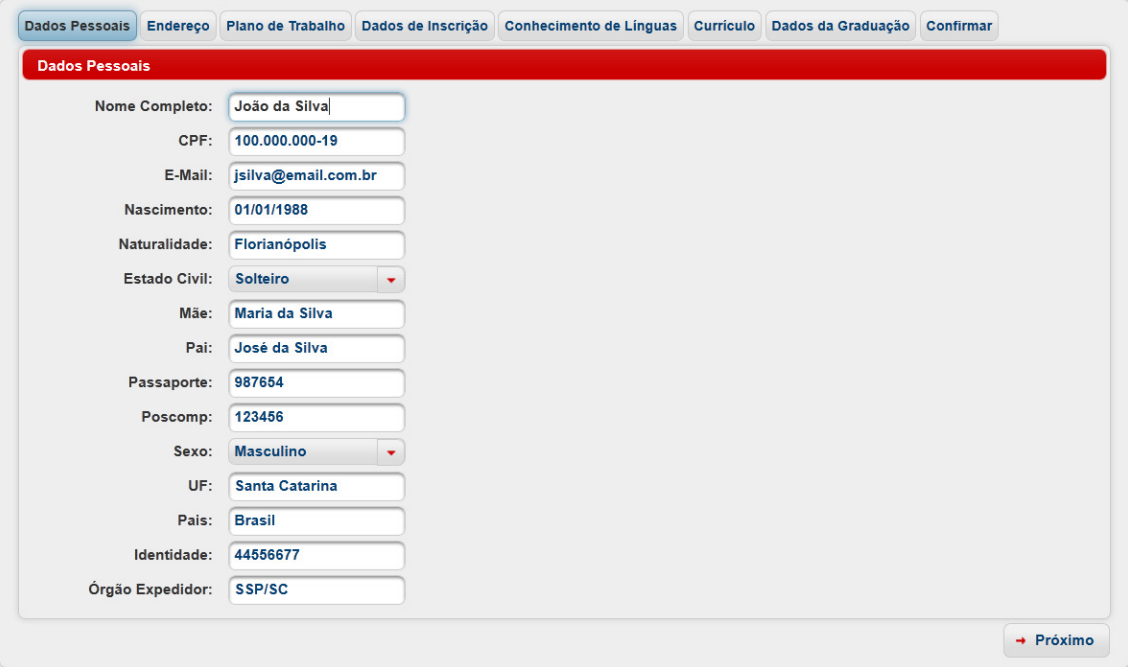

Figura 10 - Processo de inscrição do aluno.

4. Conclusões e Trabalhos Futuros

O desenvolvimento deste trabalho trouxe como aprendizado o funcionamento dos processos dentro do Programa de Pós-Graduação em Ciência da Computação, tais como o processo de inscrição de um aluno, o processo de composição das bancas de mestrado e o processo de orientação de alunos. Além disso, foram abordados sistemas de recomendação, que proporcionaram um grande aprendizado neste assunto, como os seus objetivos, as diversas formas de realizar as recomendações e coletar as informações, bem como a análise de alguns sistemas de recomendações disponíveis na internet.

O sistema desenvolvido procura centralizar as informações e os processos envolvidos durante o período de mestrado dos alunos, fazendo com que as tarefas realizadas se tornem mais fáceis de serem controladas e gerenciadas, além de serem mais ágeis e menos burocráticas, devido à ausência do uso de papéis. Através do sistema de recomendação implantado dentro da aplicação os alunos recebem um grande auxílio na decisão de um orientador, já que existem muitos professores no departamento do INE e cada um tem diversos projetos e participa de uma ou mais linhas de pesquisa. Os professores também se beneficiam com este sistema, pois recebem recomendações de alunos que mais se assemelham às suas linhas de pesquisa e projetos.

Os objetivos propostos por este trabalho foram todos concretizados, porém podemos sugerir novas funcionalidades ao sistema para torná-lo mais completo, sendo elas:

- Um sistema de Log para registrar as ações realizadas pelos usuários no sistema, como por exemplo, inclusão, alteração e exclusão de dados, além de registrar eventuais erros ocorridos no sistema;

- Melhorar a interação do aluno com o professor permitindo o compartilhamento de arquivos para melhor acompanhamento dos envolvidos no processo de defesa de dissertação;
- Inclusão de funcionalidades para a realização de backup do banco de dados via interface do sistema, além da possibilidade de poder restaurar o banco de dados em caso de perda ou comprometimento do mesmo.

5. Referências

DEMONSTRAÇÃO DOS COMPONENTES DO PRIMEFACES. Disponível em: <http://www.primefaces.org/showcase-labs/ui/home.jsf>. Acesso em: 26 de ago. 2011..

DOCUMENTAÇÃO HIBERNATE. Disponível em: <http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/>. Acesso em: 26 ago. 2011.

DOCUMENTAÇÃO POSTGRESQL. Disponível em: <http://www.postgresql.org/docs/9.0/interactive/index.html>. Acesso em: 26 de ago. 2011.

eHarmony. Disponível em: < http://www.eharmony.com.br/>. Acesso em: 2 de jul. 2011.

GONÇALVES, Edson. Segurança com Spring Security 3.0 utilizando banco de dados. Disponível em: <http://imasters.com.br/artigo/16780/seguranca/seguranca_com_spring_security_30_utilizando_banco_de_dados/>. Acesso em: 26 de ago. 2011.

JAVADOC JAVASERVER FACES. Disponível em: <http://javaserverfaces.java.net/nonav/docs/2.0/javadocs/>. Acesso em: 26 de ago. 2011.

JAVADOC HIBERNATE. Disponível em: <http://docs.jboss.org/hibernate/core/3.6/javadocs/>. Acesso em: 26 de ago. 2011.

JAVADOC PRIMEFACES. Disponível em: <http://primefaces.prime.com.tr/docs/tag/>. Acesso em: 26 de ago. 2011.

JAVADOC SPRING. Disponível em: <http://static.springsource.org/spring/docs/3.1.0.M1/javadoc-api/>. Acesso em: 26 ago. 2011.

JAVADOC SPRING SECURITY. Disponível em: <http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/index.html>. Acesso em: 26 ago. 2011.

JAVA WEB DEVELOPMENT TUTORIALS. Disponível em: <http://www.mkyong.com/>. Acesso em: 26 de ago. 2011.

MovieLens. Disponível em: < http://www.movielens.org/>. Acesso em: 5 jul. 2011.

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO. Disponível em: <http://www.ppgcc.inf.ufsc.br/>. Acesso em: 25 nov. 2010.

REATEGUI, E.B.; CAZELLA, S.C. Sistemas de Recomendação. Congresso da Sociedade Brasileira de Computação – CSBC. São Leopoldo/RS. 2005.

SISTEMA DE COORDENAÇÃO DE PROJETOS DO INE. Disponível em: <http://projetos.inf.ufsc.br/>. Acesso em: 26 ago. 2011.

SISTEMA DE RECOMENDAÇÃO. Disponível em: <http://pt.wikipedia.org/wiki/Sistema_de_recomenda%C3%A7%C3%A3o>. Acesso em: Acesso em: 26 ago. 2011.

9. Anexo B – Código Fonte

Acessos

```
package br.ufsc.inf.ppgcc.pojo;

public class Acessos implements java.io.Serializable {

    private Integer id;
    private Usuario usuario;
    private String acesso;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Usuario getUsuario() {
        return this.usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public String getAcesso() {
        return this.acesso;
    }

    public void setAcesso(String acesso) {
        this.acesso = acesso;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Acessos other = (Acessos) obj;
        if ((this.acesso == null) ? (other.acesso != null) : !this.acesso.equals(other.acesso)) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 23 * hash + (this.acesso != null ? this.acesso.hashCode() : 0);
        return hash;
```

```
    }

    @Override
    public String toString() {
        return acesso;
    }
}
```

Aluno

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class Aluno implements java.io.Serializable, Cloneable {

    private Integer id;
    private DadosAcademicos dadosAcademicos = new DadosAcademicos();
    private DadosInscricao dadosInscricao = new DadosInscricao();
    private PlanoDeTrabalho planoDeTrabalho = new PlanoDeTrabalho();
    private Endereco endereco = new Endereco();
    private Usuario usuario;
    private String nome;
    private String cpf;
    private String email;
    private Date dataNascimento;
    private String naturalidade;
    private String estadoCivil;
    private String mae;
    private String pai;
    private String atividade;
    private String passaporte;
    private Integer poscomp;
    private String sexo;
    private String estado;
    private String pais;
    private String identidade;
    private String orgaoExpedidor;
    private String correspondencia;
    private String deficiencia;
    private String inscricaoReitoria;
    private Set<TopicoDeInteresse> topicoDeInteresses = new HashSet<TopicoDeInteresse>(0);
    private Set<ConhecimentoLinguas> conhecimentosLinguas = new
HashSet<ConhecimentoLinguas>(0);
    private Set<Curriculo> curriculos = new HashSet<Curriculo>(0);
    private Orientacao orientacao;
    private Defesa defesa;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public DadosAcademicos getDadosAcademicos() {
```

```
    return this.dadosAcademicos;
}

public void setDadosAcademicos(DadosAcademicos dadosAcademicos) {
    this.dadosAcademicos = dadosAcademicos;
}

public DadosInscricao getDadosInscricao() {
    return this.dadosInscricao;
}

public void setDadosInscricao(DadosInscricao dadosInscricao) {
    this.dadosInscricao = dadosInscricao;
}

public PlanoDeTrabalho getPlanoDeTrabalho() {
    return this.planoDeTrabalho;
}

public void setPlanoDeTrabalho(PlanoDeTrabalho planoDeTrabalho) {
    this.planoDeTrabalho = planoDeTrabalho;
}

public Endereco getEndereco() {
    return this.endereco;
}

public void setEndereco(Endereco endereco) {
    this.endereco = endereco;
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

public String getNome() {
    return this.nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getCpf() {
    return this.cpf;
}

public void setCpf(String cpf) {
    this.cpf = cpf;
}

public String getEmail() {
    return this.email;
}
```

```
public void setEmail(String email) {
    this.email = email;
}

public Date getDataNascimento() {
    return this.dataNascimento;
}

public void setDataNascimento(Date dataNascimento) {
    this.dataNascimento = dataNascimento;
}

public String getNaturalidade() {
    return this.naturalidade;
}

public void setNaturalidade(String naturalidade) {
    this.naturalidade = naturalidade;
}

public String getEstadoCivil() {
    return this.estadoCivil;
}

public void setEstadoCivil(String estadoCivil) {
    this.estadoCivil = estadoCivil;
}

public String getMae() {
    return this.mae;
}

public void setMae(String mae) {
    this.mae = mae;
}

public String getPai() {
    return this.pai;
}

public void setPai(String pai) {
    this.pai = pai;
}

public String getAtividade() {
    return this.atividade;
}

public void setAtividade(String atividade) {
    this.atividade = atividade;
}

public String getPassaporte() {
    return this.passaporte;
}

public void setPassaporte(String passaporte) {
    this.passaporte = passaporte;
}
```

```
public Integer getPoscomp() {
    return this.poscomp;
}

public void setPoscomp(Integer poscomp) {
    this.poscomp = poscomp;
}

public String getSexo() {
    return this.sexo;
}

public void setSexo(String sexo) {
    this.sexo = sexo;
}

public String getEstado() {
    return this.estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getPais() {
    return this.pais;
}

public void setPais(String pais) {
    this.pais = pais;
}

public String getIdentidade() {
    return this.identidade;
}

public void setIdentidade(String identidade) {
    this.identidade = identidade;
}

public String getOrgaoExpedidor() {
    return this.orgaoExpedidor;
}

public void setOrgaoExpedidor(String orgaoExpedidor) {
    this.orgaoExpedidor = orgaoExpedidor;
}

public String getCorrespondencia() {
    return this.correspondencia;
}

public void setCorrespondencia(String correspondencia) {
    this.correspondencia = correspondencia;
}

public String getDeficiencia() {
    return this.deficiencia;
```

```
}

public void setDeficiencia(String deficiencia) {
    this.deficiencia = deficiencia;
}

public String getInscricaoReitoria() {
    return inscricaoReitoria;
}

public void setInscricaoReitoria(String inscricaoReitoria) {
    this.inscricaoReitoria = inscricaoReitoria;
}

public Set<TopicoDeInteresse> getTopicoDeInteresses() {
    return this.topicoDeInteresses;
}

public void setTopicoDeInteresses(Set<TopicoDeInteresse> topicoDeInteresses) {
    this.topicoDeInteresses = topicoDeInteresses;
}

public Orientacao getOrientacao() {
    return orientacao;
}

public void setOrientacao(Orientacao orientacao) {
    this.orientacao = orientacao;
}

public Set<ConhecimentoLinguas> getConhecimentosLinguas() {
    return this.conhecimentosLinguas;
}

public void setConhecimentosLinguas(Set<ConhecimentoLinguas> conhecimentosLinguas) {
    this.conhecimentosLinguas = conhecimentosLinguas;
}

public Defesa getDefesa() {
    return defesa;
}

public void setDefesa(Defesa defesa) {
    this.defesa = defesa;
}

public Set<Curriculo> getCurriculos() {
    return this.curriculos;
}

public void setCurriculos(Set<Curriculo> curriculos) {
    this.curriculos = curriculos;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
```

```
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Aluno other = (Aluno) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 23 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }

    @Override
    public String toString() {
        return nome;
    }

    @Override
    public Aluno clone() {
        try {
            return (Aluno) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
    }
}
```

Banca

```
package br.ufsc.inf.ppgcc.pojo;

public class Banca implements java.io.Serializable {

    private BancaId id;
    private Defesa defesa;
    private Professor professor;
    private short status;
    /*
     * Status
     */
    public static final short PENDENTE = 0;
    public static final short RECUSADO_COMITE = 10;
    public static final short ACEITO_COMITE = 20;
    public static final short RECUSADO_PROFESSOR = 30;
    public static final short ACEITO_PROFESSOR = 40;

    public BancaId getId() {
        return this.id;
    }

    public void setId(BancaId id) {
        this.id = id;
    }
```

```
public Defesa getDefesa() {
    return this.defesa;
}

public void setDefesa(Defesa defesa) {
    this.defesa = defesa;
}

public Professor getProfessor() {
    return this.professor;
}

public void setProfessor(Professor professor) {
    this.professor = professor;
}

public short getStatus() {
    return status;
}

public void setStatus(short status) {
    this.status = status;
}

public String getStatusProfessor() {
    switch (status) {
        case PENDENTE:
            return "Aguardando análise";
        case RECUSADO_COMITE:
            return "Recusado pelo comitê";
        case ACEITO_COMITE:
            return "Aceito pelo comitê";
        case RECUSADO_PROFESSOR:
            return "Recusado pelo professor";
        case ACEITO_PROFESSOR:
            return "Aceito pelo professor";
    }
    return "";
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Banca other = (Banca) obj;
    if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 19 * hash + (this.id != null ? this.id.hashCode() : 0);
```

```
        return hash;
    }
}
```

BancaId

```
package br.ufsc.inf.ppgcc.pojo;

public class BancaId implements java.io.Serializable {

    private int defesa;
    private int professor;

    public int getDefesa() {
        return this.defesa;
    }

    public void setDefesa(int defesa) {
        this.defesa = defesa;
    }

    public int getProfessor() {
        return this.professor;
    }

    public void setProfessor(int professor) {
        this.professor = professor;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final BancaId other = (BancaId) obj;
        if (this.defesa != other.defesa) {
            return false;
        }
        if (this.professor != other.professor) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 67 * hash + this.defesa;
        hash = 67 * hash + this.professor;
        return hash;
    }
}
```

ConhecimentoLinguas

```
package br.ufsc.inf.ppgcc.pojo;

public class ConhecimentoLinguas implements java.io.Serializable {

    private Integer id;
    private Aluno aluno;
    private String lingua;
    private String leitura;
    private String redacao;
    private String traducao;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Aluno getAluno() {
        return this.aluno;
    }

    public void setAluno(Aluno aluno) {
        this.aluno = aluno;
    }

    public String getLingua() {
        return lingua;
    }

    public void setLingua(String lingua) {
        this.lingua = lingua;
    }

    public String getLeitura() {
        return this.leitura;
    }

    public void setLeitura(String leitura) {
        this.leitura = leitura;
    }

    public String getRedacao() {
        return this.redacao;
    }

    public void setRedacao(String redacao) {
        this.redacao = redacao;
    }

    public String getTraducao() {
        return this.traducao;
    }

    public void setTraducao(String traducao) {
        this.traducao = traducao;
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final ConhecimentoLinguas other = (ConhecimentoLinguas) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        if ((this.lingua == null) ? (other.lingua != null) : !this.lingua.equals(other.lingua)) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 29 * hash + (this.id != null ? this.id.hashCode() : 0);
        hash = 29 * hash + (this.lingua != null ? this.lingua.hashCode() : 0);
        return hash;
    }
}
```

Curriculo

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.Date;

public class Curriculo implements java.io.Serializable, Cloneable {

    private Integer id;
    private Aluno aluno;
    private String instituicao;
    private String localNaInstituicao;
    private String cidade;
    private String estado;
    private String pais;
    private String funcao;
    private String atividades;
    private Integer cargaHoraria;
    private Date dataInicio;
    private Date dataFim;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Aluno getAluno() {
        return this.aluno;
    }
```

```
public void setAluno(Aluno aluno) {
    this.aluno = aluno;
}

public String getInstituicao() {
    return this.instituicao;
}

public void setInstituicao(String instituicao) {
    this.instituicao = instituicao;
}

public String getLocalNaInstituicao() {
    return this.localNaInstituicao;
}

public void setLocalNaInstituicao(String localNaInstituicao) {
    this.localNaInstituicao = localNaInstituicao;
}

public String getCidade() {
    return this.cidade;
}

public void setCidade(String cidade) {
    this.cidade = cidade;
}

public String getEstado() {
    return this.estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getPais() {
    return this.pais;
}

public void setPais(String pais) {
    this.pais = pais;
}

public String getFuncao() {
    return this.funcao;
}

public void setFuncao(String funcao) {
    this.funcao = funcao;
}

public String getAtividades() {
    return this.atividades;
}

public void setAtividades(String atividades) {
    this.atividades = atividades;
```

```
    }

    public Integer getCargaHoraria() {
        return this.cargaHoraria;
    }

    public void setCargaHoraria(Integer cargaHoraria) {
        this.cargaHoraria = cargaHoraria;
    }

    public Date getDataInicio() {
        return this.dataInicio;
    }

    public void setDataInicio(Date dataInicio) {
        this.dataInicio = dataInicio;
    }

    public Date getDataFim() {
        return this.dataFim;
    }

    public void setDataFim(Date dataFim) {
        this.dataFim = dataFim;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Curriculo other = (Curriculo) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        if ((this.instituicao == null) ? (other.instituicao != null) :
!this.instituicao.equals(other.instituicao)) {
            return false;
        }
        if ((this.localNaInstituicao == null) ? (other.localNaInstituicao != null) :
!this.localNaInstituicao.equals(other.localNaInstituicao)) {
            return false;
        }
        if (this.dataInicio != other.dataInicio && (this.dataInicio == null ||
!this.dataInicio.equals(other.dataInicio))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 89 * hash + (this.id != null ? this.id.hashCode() : 0);
        hash = 89 * hash + (this.instituicao != null ? this.instituicao.hashCode() : 0);
        hash = 89 * hash + (this.localNaInstituicao != null ? this.localNaInstituicao.hashCode() : 0);
```

```
        hash = 89 * hash + (this.dataInicio != null ? this.dataInicio.hashCode() : 0);
        return hash;
    }

    @Override
    public Curriculo clone() {
        try {
            return (Curriculo) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
    }

    public Curriculo copy(Curriculo curriculo) {
        instituicao = curriculo.getInstituicao();
        localNaInstituicao = curriculo.getLocalNaInstituicao();
        cidade = curriculo.getCidade();
        estado = curriculo.getEstado();
        pais = curriculo.getPais();
        funcao = curriculo.getFuncao();
        atividades = curriculo.getAtividades();
        cargaHoraria = curriculo.getCargaHoraria();
        dataInicio = curriculo.getDataInicio();
        dataFim = curriculo.getDataFim();

        return this;
    }
}
```

DadosAcademicos

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class DadosAcademicos implements java.io.Serializable {

    private Integer id;
    private String instituicao;
    private String curso;
    private String cidade;
    private String estado;
    private String pais;
    private Date inicio;
    private Date conclusao;
    private String trabalho;
    private Double indice;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getInstituicao() {
```

```
    return this.instituicao;
}

public void setInstituicao(String instituicao) {
    this.instituicao = instituicao;
}

public String getCurso() {
    return this.curso;
}

public void setCurso(String curso) {
    this.curso = curso;
}

public String getCidade() {
    return this.cidade;
}

public void setCidade(String cidade) {
    this.cidade = cidade;
}

public String getEstado() {
    return this.estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getPais() {
    return this.pais;
}

public void setPais(String pais) {
    this.pais = pais;
}

public Date getInicio() {
    return this.inicio;
}

public void setInicio(Date inicio) {
    this.inicio = inicio;
}

public Date getConclusao() {
    return this.conclusao;
}

public void setConclusao(Date conclusao) {
    this.conclusao = conclusao;
}

public String getTrabalho() {
    return this.trabalho;
}
```

```
    public void setTrabalho(String trabalho) {
        this.trabalho = trabalho;
    }

    public Double getIndice() {
        return this.indice;
    }

    public void setIndice(Double indice) {
        this.indice = indice;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final DadosAcademicos other = (DadosAcademicos) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 29 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

DadosInscricao

```
package br.ufsc.inf.ppgcc.pojo;

public class DadosInscricao implements java.io.Serializable {

    private Integer id;
    private Professor professor;
    private LinhaDePesquisa linhaDePesquisa;
    private String projeto;
    private boolean bolsa;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Professor getProfessor() {
        return this.professor;
    }
```

```
    public void setProfessor(Professor professor) {
        this.professor = professor;
    }

    public LinhaDePesquisa getLinhaDePesquisa() {
        return this.linhaDePesquisa;
    }

    public void setLinhaDePesquisa(LinhaDePesquisa linhaDePesquisa) {
        this.linhaDePesquisa = linhaDePesquisa;
    }

    public String getProjeto() {
        return this.projeto;
    }

    public void setProjeto(String projeto) {
        this.projeto = projeto;
    }

    public boolean isBolsa() {
        return this.bolsa;
    }

    public void setBolsa(boolean bolsa) {
        this.bolsa = bolsa;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final DadosInscricao other = (DadosInscricao) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 47 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

Defesa

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
```

```
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Defesa implements java.io.Serializable, Cloneable {

    private Integer idAluno;
    private Aluno aluno;
    private Date data;
    private String local;
    private Set<Banca> bancas = new HashSet<Banca>(0);
    private boolean concluida;
    private boolean mostrar;

    public Integer getIdAluno() {
        return this.idAluno;
    }

    public void setIdAluno(Integer idAluno) {
        this.idAluno = idAluno;
    }

    public Aluno getAluno() {
        return this.aluno;
    }

    public void setAluno(Aluno aluno) {
        this.aluno = aluno;
    }

    public Date getData() {
        return this.data;
    }

    public void setData(Date data) {
        this.data = data;
    }

    public String getLocal() {
        return local;
    }

    public void setLocal(String local) {
        this.local = local;
    }

    public Set<Banca> getBancas() {
        return bancas;
    }

    public void setBancas(Set<Banca> bancas) {
        this.bancas = bancas;
    }

    public List<Banca> getBancasList() {
        return new ArrayList<Banca>(bancas);
    }

    public boolean isConcluida() {
```

```
        return concluida;
    }

    public void setConcluida(boolean concluida) {
        this.concluida = concluida;
    }

    public boolean isMostrar() {
        return mostrar;
    }

    public void setMostrar(boolean mostrar) {
        this.mostrar = mostrar;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Defesa other = (Defesa) obj;
        if (this.idAluno != other.idAluno && (this.idAluno == null ||
!this.idAluno.equals(other.idAluno))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 59 * hash + (this.idAluno != null ? this.idAluno.hashCode() : 0);
        return hash;
    }

    @Override
    public Defesa clone() {
        try {
            return (Defesa) super.clone();
        } catch (CloneNotSupportedException ex) {
            return null;
        }
    }

    public void copy(Defesa defesa) {
        aluno = defesa.getAluno();
        data = defesa.getData();
        local = defesa.getLocal();
        bancas = defesa.getBancas();
        concluida = defesa.isConcluida();
        mostrar = defesa.isMostrar();
    }
}
```

Endereco

```
package br.ufsc.inf.ppgcc.pojo;

public class Endereco implements java.io.Serializable {

    private Integer id;
    private String logradouro;
    private String complemento;
    private String bairro;
    private String cep;
    private String cidade;
    private String estado;
    private String pais;
    private String telefone;
    private String fax;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getLogradouro() {
        return this.logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    public String getComplemento() {
        return this.complemento;
    }

    public void setComplemento(String complemento) {
        this.complemento = complemento;
    }

    public String getBairro() {
        return this.bairro;
    }

    public void setBairro(String bairro) {
        this.bairro = bairro;
    }

    public String getCep() {
        return this.cep;
    }

    public void setCep(String cep) {
        this.cep = cep;
    }

    public String getCidade() {
        return this.cidade;
    }
```

```
    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public String getEstado() {
        return this.estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    public String getPais() {
        return this.pais;
    }

    public void setPais(String pais) {
        this.pais = pais;
    }

    public String getTelefone() {
        return this.telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public String getFax() {
        return this.fax;
    }

    public void setFax(String fax) {
        this.fax = fax;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Endereco other = (Endereco) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 97 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

Grupo

```
package br.ufsc.inf.ppgcc.pojo;

public class Grupo implements java.io.Serializable, Cloneable {

    private Integer id;
    private String nome;
    private Integer numeroDeOrientacoes;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Integer getNumeroDeOrientacoes() {
        return numeroDeOrientacoes;
    }

    public void setNumeroDeOrientacoes(Integer numeroDeOrientacoes) {
        this.numeroDeOrientacoes = numeroDeOrientacoes;
    }

    @Override
    public String toString() {
        return nome;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Grupo other = (Grupo) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 23 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
```

```
    }

    @Override
    public Grupo clone() {
        try {
            return (Grupo) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
    }

    public void copy(Grupo grupo) {
        nome = grupo.getNome();
        numeroDeOrientacoes = grupo.getNumeroDeOrientacoes();
    }
}
```

LinhaDePesquisa

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class LinhaDePesquisa implements java.io.Serializable, Cloneable {

    private Integer id;
    private Professor coordenador;
    private String objetivos;
    private String titulo;
    private Set<Professor> professores = new HashSet<Professor>(0);

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Professor getCoordenador() {
        return this.coordenador;
    }

    public void setCoordenador(Professor coordenador) {
        this.coordenador = coordenador;
    }

    public String getObjetivos() {
        return this.objetivos;
    }

    public void setObjetivos(String objetivos) {
        this.objetivos = objetivos;
    }

    public String getTitulo() {
```

```
    return titulo;
}

public void setTitulo(String titulo) {
    this.titulo = titulo;
}

public Set<Professor> getProfessores() {
    return this.professores;
}

public void setProfessores(Set<Professor> professores) {
    this.professores = professores;
}

public List<Professor> getProfessoresList() {
    return new ArrayList<Professor>(this.getProfessores());
}

public int getProfessoresSize() {
    return professores.size();
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final LinhaDePesquisa other = (LinhaDePesquisa) obj;
    if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 59 * hash + (this.id != null ? this.id.hashCode() : 0);
    return hash;
}

@Override
public String toString() {
    return titulo;
}

@Override
public LinhaDePesquisa clone() {
    try {
        return (LinhaDePesquisa) super.clone();
    } catch (CloneNotSupportedException ex) {
    }
    return null;
}
```

```
    public void copy(LinhaDePesquisa linhaDePesquisa) {
        coordenador = linhaDePesquisa.getCoordenador();
        titulo = linhaDePesquisa.getTitulo();
        objetivos = linhaDePesquisa.getObjetivos();
        professores = linhaDePesquisa.getProfessores();
    }
}
```

Orientacao

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.Date;

public class Orientacao implements java.io.Serializable {

    private OrientacaoId id;
    private Aluno aluno;
    private Professor professor;
    private Date dataInicio;
    private boolean bolsista;
    private Boolean convite;
    private Boolean aceito;
    private boolean concluida;

    public OrientacaoId getId() {
        return this.id;
    }

    public void setId(OrientacaoId id) {
        this.id = id;
    }

    public Aluno getAluno() {
        return this.aluno;
    }

    public void setAluno(Aluno aluno) {
        this.aluno = aluno;
    }

    public Professor getProfessor() {
        return this.professor;
    }

    public void setProfessor(Professor professor) {
        this.professor = professor;
    }

    public Date getDataInicio() {
        return this.dataInicio;
    }

    public void setDataInicio(Date dataInicio) {
        this.dataInicio = dataInicio;
    }

    public boolean isBolsista() {
        return this.bolsista;
```

```
    }

    public void setBolsista(boolean bolsista) {
        this.bolsista = bolsista;
    }

    public Boolean getAceito() {
        return aceito;
    }

    public void setAceito(Boolean aceito) {
        this.aceito = aceito;
    }

    public Boolean getConvite() {
        return convite;
    }

    public void setConvite(Boolean convite) {
        this.convite = convite;
    }

    public boolean isConcluida() {
        return concluida;
    }

    public void setConcluida(boolean concluida) {
        this.concluida = concluida;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Orientacao other = (Orientacao) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 37 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

OrientacaoId

```
package br.ufsc.inf.ppgcc.pojo;

public class OrientacaoId implements java.io.Serializable {
```

```
    private int aluno;
    private int professor;

    public OrientacaoId() {
    }

    public OrientacaoId(int aluno, int professor) {
        this.aluno = aluno;
        this.professor = professor;
    }

    public int getAluno() {
        return this.aluno;
    }

    public void setAluno(int aluno) {
        this.aluno = aluno;
    }

    public int getProfessor() {
        return this.professor;
    }

    public void setProfessor(int professor) {
        this.professor = professor;
    }

    @Override
    public boolean equals(Object other) {
        if ((this == other)) {
            return true;
        }
        if ((other == null)) {
            return false;
        }
        if (!(other instanceof OrientacaoId)) {
            return false;
        }
        OrientacaoId castOther = (OrientacaoId) other;

        return (this.getAluno() == castOther.getAluno())
                && (this.getProfessor() == castOther.getProfessor());
    }

    @Override
    public int hashCode() {
        int result = 17;

        result = 37 * result + this.getAluno();
        result = 37 * result + this.getProfessor();
        return result;
    }
}
```

PlanoDeTrabalho

```
package br.ufsc.inf.ppgcc.pojo;

public class PlanoDeTrabalho implements java.io.Serializable {
```

```
    private Integer id;
    private String titulo;
    private String plano;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getTitulo() {
        return this.titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getPlano() {
        return this.plano;
    }

    public void setPlano(String plano) {
        this.plano = plano;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final PlanoDeTrabalho other = (PlanoDeTrabalho) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 97 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

Professor

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
```

```
import java.util.List;
import java.util.Set;

public class Professor implements java.io.Serializable, Cloneable {

    private Integer id;
    private Usuario usuario;
    private Grupo grupo;
    private String nome;
    private String email;
    private String website;
    private boolean professorINE;
    private Set<Orientacao> orientacoes = new HashSet<Orientacao>(0);
    private Set<Defesa> defesas = new HashSet<Defesa>(0);
    private Set<TopicoDeInteresse> topicoDeInteresses = new HashSet<TopicoDeInteresse>(0);
    private LinhaDePesquisa coordena;
    private Set<LinhaDePesquisa> linhasDePesquisa = new HashSet<LinhaDePesquisa>(0);
    private Set<Banca> bancas = new HashSet<Banca>(0);
    private Set<Projeto> projetos = new HashSet<Projeto>(0);

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Grupo getGrupo() {
        return grupo;
    }

    public void setGrupo(Grupo grupo) {
        this.grupo = grupo;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public String getNome() {
        return this.nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEmail() {
        return this.email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```

```
public String getWebsite() {
    return this.website;
}

public void setWebsite(String website) {
    this.website = website;
}

public boolean isProfessorINE() {
    return professorINE;
}

public void setProfessorINE(boolean professorINE) {
    this.professorINE = professorINE;
}

public Set<Orientacao> getOrientacoes() {
    return this.orientacoes;
}

public void setOrientacoes(Set<Orientacao> orientacoes) {
    this.orientacoes = orientacoes;
}

public List<Orientacao> getOrientacoesList() {
    return new ArrayList<Orientacao>(this.getOrientacoes());
}

public Set<Defesa> getDefesas() {
    return this.defesas;
}

public void setDefesas(Set<Defesa> defesas) {
    this.defesas = defesas;
}

public Set<TopicoDeInteresse> getTopicoDeInteresses() {
    return this.topicoDeInteresses;
}

public void setTopicoDeInteresses(Set<TopicoDeInteresse> topicoDeInteresses) {
    this.topicoDeInteresses = topicoDeInteresses;
}

public LinhaDePesquisa getCoordena() {
    return coordena;
}

public void setCoordena(LinhaDePesquisa coordena) {
    this.coordena = coordena;
}

public Set<LinhaDePesquisa> getLinhasDePesquisa() {
    return linhasDePesquisa;
}

public void setLinhasDePesquisa(Set<LinhaDePesquisa> linhasDePesquisa) {
    this.linhasDePesquisa = linhasDePesquisa;
```

```
}

public String getLinhasDePesquisaString() {
    String linhas = "";
    Iterator<LinhaDePesquisa> it = linhasDePesquisa.iterator();
    while (it.hasNext()) {
        linhas += it.next().getTitulo();
        if (it.hasNext()) {
            linhas += ", ";
        }
    }
    return linhas;
}

public Set<Banca> getBancas() {
    return this.bancas;
}

public void setBancas(Set<Banca> bancas) {
    this.bancas = bancas;
}

public List<Banca> getBancasList() {
    return new ArrayList<Banca>(bancas);
}

public Set<Projeto> getProjetos() {
    return projetos;
}

public void setProjetos(Set<Projeto> projetos) {
    this.projetos = projetos;
}

public List<Projeto> getProjetosList() {
    return new ArrayList<Projeto>(projetos);
}

public boolean isCoordenador() {
    return coordena != null;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Professor other = (Professor) obj;
    if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
```

```
        int hash = 7;
        hash = 29 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }

    @Override
    public String toString() {
        return nome;
    }

    @Override
    public Professor clone() {
        try {
            return (Professor) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
    }

    public void copy(Professor professor) {
        nome = professor.getNome();
        email = professor.getEmail();
        website = professor.getWebsite();
        linhasDePesquisa = professor.getLinhasDePesquisa();
        grupo = professor.getGrupo();
    }
}
```

Projeto

```
package br.ufsc.inf.ppgcc.pojo;

public class Projeto implements java.io.Serializable, Cloneable {

    private Integer id;
    private String titulo;
    private String descricao;
    private Professor professor;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Professor getProfessor() {
        return professor;
    }

    public void setProfessor(Professor professor) {
        this.professor = professor;
    }

    public String getTitulo() {
        return titulo;
    }
```

```
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getDescricaoCurta() {
        return descricao.length() > 100 ? descricao.substring(0, 100) + "..." : descricao;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    @Override
    public Projeto clone() {
        try {
            return (Projeto) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
    }

    public void copy(Projeto projeto) {
        titulo = projeto.getTitulo();
        professor = projeto.getProfessor();
        descricao = projeto.getDescricao();
    }
}
```

Usuario

```
package br.ufsc.inf.ppgcc.pojo;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class Usuario implements java.io.Serializable, Cloneable {

    private Integer id;
    private String login;
    private String senha;
    private boolean habilitado = true;
    private Set<Acessos> acessos = new HashSet<Acessos>(0);
    private Aluno aluno;
    private Professor professor;

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getLogin() {
```

```
    return this.login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return this.senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

public Aluno getAluno() {
    return aluno;
}

public void setAluno(Aluno aluno) {
    this.aluno = aluno;
}

public boolean isHabilitado() {
    return habilitado;
}

public void setHabilitado(boolean habilitado) {
    this.habilitado = habilitado;
}

public Set<Acessos> getAcessos() {
    return acessos;
}

public void setAcessos(Set<Acessos> acessos) {
    this.acessos = acessos;
}

public Professor getProfessor() {
    return professor;
}

public void setProfessor(Professor professor) {
    this.professor = professor;
}

public String getAcessoString() {
    String a = "";
    String temp;
    Iterator<Acessos> it = acessos.iterator();
    while (it.hasNext()) {
        temp = it.next().getAcesso().replace("ROLE_", "");
        temp = temp.substring(0, 1) + temp.substring(1).toLowerCase();
        a += temp;
        if (it.hasNext()) {
            a += ", ";
        }
    }
```

```
        return a;
    }

    public boolean isAssociadoAProfessor() {
        return professor != null;
    }

    public boolean isAssociadoACoordenador() {
        return this.isAssociadoAProfessor() && professor.isCoordenador();
    }

    public boolean isAssociadoAAluno() {
        return aluno != null;
    }

    public boolean isSemAssociacao() {
        return professor == null && aluno == null;
    }

    public String getNomeAssociado() {
        return this.isAssociadoAAluno() ? aluno.getNome() : this.isAssociadoAProfessor() ?
professor.getNome() : "";
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Usuario other = (Usuario) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 67 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }

    @Override
    public String toString() {
        return login;
    }

    @Override
    public Usuario clone() {
        try {
            return (Usuario) super.clone();
        } catch (CloneNotSupportedException ex) {
        }
        return null;
```

```
    }

    public void copy(Usuario usuario) {
        login = usuario.getLogin();
        habilitado = usuario.isHabilitado();
        acessos = usuario.getAcessos();
        aluno = usuario.getAluno();
        professor = usuario.getProfessor();
    }
}
```

AcessosDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.AcessosDAO;
import br.ufsc.inf.ppgcc.pojo.Acessos;
import br.ufsc.inf.ppgcc.pojo.Usuario;
import br.ufsc.inf.ppgcc.security.Acesso;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Restrictions;
import org.springframework.dao.support.DataAccessUtils;
import org.springframework.stereotype.Repository;

@Repository
public class AcessosDAOImpl extends CustomHibernateDaoSupport implements AcessosDAO
{

    public void deletar(Acessos acessos) {
        if (acessos.getId() == null) {
            getHibernateTemplate().delete(getAcessos(acessos.getUsuario(),
acessos.getAcesso()));
        } else {
            getHibernateTemplate().delete(acessos);
        }
    }

    public Acessos getAcessos(Usuario usuario, String acesso) {
        return (Acessos)
DataAccessUtils.uniqueResult(getHibernateTemplate().findByCriteria(DetachedCriteria.forClass
(Acessos.class).
                add(Restrictions.and(
                Restrictions.eq("usuario", usuario),
                Restrictions.eq("acesso", acesso)))));
    }

    public void alterarPresidenteComiteBanca(Usuario usuario) throws Exception {
        Session session = getSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();

            Query query = session.createQuery("delete from Acessos a where a.acesso = ?");
            query.setString(0, Acesso.COMITE_BANCA.getAuthority());
            query.executeUpdate();
```

```
            Acessos acessos = new Acessos();
            acessos.setUsuario(usuario);
            acessos.setAcesso(Acesso.COMITE_BANCA.getAuthority());
            session.save(acessos);

            tx.commit();
        } catch (Exception e) {
            if (tx != null) {
                tx.rollback();
            }
            throw e;
        } finally {
            session.close();
        }
    }
}
```

AlunoDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.AlunoDAO;
import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.MatchMode;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

@Repository
public class AlunoDAOImpl extends CustomHibernateDaoSupport implements AlunoDAO {

    public void save(Aluno aluno) {
        getHibernateTemplate().saveOrUpdate(aluno);
    }

    public List<Aluno> getByNome(String nome) {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Aluno.class)
                .add(Restrictions.like("nome", nome, MatchMode.ANYWHERE).ignoreCase())
                .addOrder(Order.asc("nome")));
    }

    public List<Aluno> getSemUsuario() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Aluno.class)
                .add(Restrictions.isNull("usuario"))
                .addOrder(Order.asc("nome")));
    }

    public Aluno getById(int id) {
```

```
        return getHibernateTemplate().get(Aluno.class, id);
    }

    public List<Aluno> getComProfessor(Professor professor) {
        return getHibernateTemplate().find
                ("SELECT a FROM Aluno a "
                + "LEFT JOIN FETCH a.dadosInscricao d "
                + "LEFT JOIN FETCH a.orientacao o "
                + "LEFT JOIN FETCH a.planoDeTrabalho "
                + "WHERE o IS NULL "
                + "AND d.professor = ?"
                , professor);
    }

    public List<Aluno> getComLinhaDePesquisa(Set<LinhaDePesquisa> linhasDePesquisa) {
        String query = "SELECT a FROM Aluno a "
                + "LEFT JOIN FETCH a.dadosInscricao d "
                + "LEFT JOIN FETCH a.orientacao o "
                + "LEFT JOIN FETCH a.planoDeTrabalho "
                + "WHERE o IS NULL "
                + "AND d.professor IS NULL "
                + "AND (";

        List<LinhaDePesquisa> list = new ArrayList<LinhaDePesquisa>();
        Iterator<LinhaDePesquisa> it = linhasDePesquisa.iterator();

        while (it.hasNext()) {
            query += " OR d.linhaDePesquisa = ?";
            list.add(it.next());
        }

        query = query.replaceFirst(" OR ", "") + ")";

        return getHibernateTemplate().find(query, list.toArray());
    }

    public List<Aluno> getComProfessorOuLinhaDePesquisa(Professor professor,
Set<LinhaDePesquisa> linhasDePesquisa) {
        String query = "SELECT a FROM Aluno a "
                + "LEFT JOIN FETCH a.dadosInscricao d "
                + "LEFT JOIN FETCH a.orientacao o "
                + "LEFT JOIN FETCH a.planoDeTrabalho "
                + "WHERE o IS NULL "
                + "AND ( d.professor = ? "
                + "OR ";

        List<Object> list = new ArrayList<Object>();
        Iterator<LinhaDePesquisa> it = linhasDePesquisa.iterator();

        list.add(professor);

        while (it.hasNext()) {
            query += " OR d.linhaDePesquisa = ?";
            list.add(it.next());
        }

        query = query.replaceFirst(" OR ", "") + ")";

        return getHibernateTemplate().find(query, list.toArray());
```

```
    }

}
```

BancaDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.BancaDAO;
import br.ufsc.inf.ppgcc.pojo.Banca;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.hibernate.FetchMode;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

@Repository
public class BancaDAOImpl extends CustomHibernateDaoSupport implements BancaDAO {

    public List<Banca> getBancasByStatus(short status) {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Banca.class)
                .createAlias("defesa.aluno", "a")
                .setFetchMode("defesa", FetchMode.JOIN)
                .setFetchMode("defesa.aluno", FetchMode.JOIN)
                .setFetchMode("professor", FetchMode.JOIN)
                .add(Restrictions.eq("status", status))
                .addOrder(Order.asc("a.nome")));
    }

    public List<Banca> getBancasByStatus(short status, Professor professor) {
        return getHibernateTemplate().findByNamedParam("SELECT b FROM Banca b "
                + "left join fetch b.defesa d "
                + "left join fetch b.professor p "
                + "left join fetch d.aluno a "
                + "left join fetch a.orientacao o "
                + "left join fetch o.professor "
                + "WHERE p.id = :p "
                + "AND b.status = :s "
                + "ORDER BY a.nome",
                new String[]{"p", "s"}, new Object[]{professor.getId(), status});
    }

    public void save(Banca banca) {
        getHibernateTemplate().save(banca);
    }

    public void update(Banca banca) {
        getHibernateTemplate().update(banca);
    }

    public void delete(Banca banca) {
        getHibernateTemplate().delete(banca);
    }
}
```

DefesaDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.DefesaDAO;
import br.ufsc.inf.ppgcc.pojo.Defesa;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.springframework.dao.support.DataAccessUtils;
import org.springframework.stereotype.Repository;

@Repository
public class DefesaDAOImpl extends CustomHibernateDaoSupport implements DefesaDAO {

    public void save(Defesa defesa) {
        getHibernateTemplate().saveOrUpdate(defesa);
    }

    public List<Defesa> getAll() {
        return getHibernateTemplate().find("SELECT d FROM Defesa d LEFT JOIN FETCH
d.aluno ORDER BY d.data");
    }

    public List<Defesa> getByMostrar() {
        return getHibernateTemplate().find("SELECT d FROM Defesa d LEFT JOIN FETCH
d.aluno WHERE d.mostrar = true ORDER BY d.data");
    }

    public Defesa getById(Integer id) {
        return (Defesa) DataAccessUtils.uniqueResult(getHibernateTemplate()
                .find("SELECT d FROM Defesa d LEFT JOIN FETCH d.aluno WHERE d.idAluno = ?",
id));
    }
}
```

GrupoDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.GrupoDAO;
import br.ufsc.inf.ppgcc.pojo.Grupo;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Order;
import org.springframework.stereotype.Repository;

@Repository
public class GrupoDAOImpl extends CustomHibernateDaoSupport implements GrupoDAO {

    public void save(Grupo grupo) {
        if (grupo.getId() == null) {
            getHibernateTemplate().save(grupo);
        } else {
            getHibernateTemplate().update(grupo);
        }
    }
```

```
    public void delete(Grupo grupo) {
        getHibernateTemplate().delete(grupo);
    }

    public List<Grupo> getAll() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Grupo.class)
                .addOrder(Order.asc("nome")));
    }

    public Grupo getById(Integer id) {
        return getHibernateTemplate().get(Grupo.class, id);
    }
}
```

LinhaDePesquisaDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.LinhaDePesquisaDAO;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.FetchMode;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.MatchMode;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

@Repository
public class LinhaDePesquisaDAOImpl extends CustomHibernateDaoSupport implements
LinhaDePesquisaDAO {

    public void save(LinhaDePesquisa linhaDePesquisa) {
        if (linhaDePesquisa.getId() == null) {
            getHibernateTemplate().save(linhaDePesquisa);
        } else {
            getHibernateTemplate().update(linhaDePesquisa);
        }
    }

    public void delete(LinhaDePesquisa linhaDePesquisa) {
        getHibernateTemplate().delete(linhaDePesquisa);
    }

    public List<LinhaDePesquisa> getByTitulo(String titulo) {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(LinhaDePesquisa.class)
                .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY)
                .setFetchMode("coordenador", FetchMode.JOIN)
                .setFetchMode("professores", FetchMode.JOIN)
                .add(Restrictions.like("titulo", titulo, MatchMode.ANYWHERE).ignoreCase())
                .addOrder(Order.asc("titulo")));
    }

    public LinhaDePesquisa getById(int id) {
        return getHibernateTemplate().get(LinhaDePesquisa.class, id);
```

```
    }

    public List<LinhaDePesquisa> getAll() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(LinhaDePesquisa.class)
                .addOrder(Order.asc("titulo")));
    }

    public List<LinhaDePesquisa> getAllFetch() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(LinhaDePesquisa.class)
                .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY)
                .setFetchMode("coordenador", FetchMode.JOIN)
                .setFetchMode("coordenador.linhasDePesquisa", FetchMode.JOIN)
                .setFetchMode("professores", FetchMode.JOIN)
                .addOrder(Order.asc("titulo")));
    }

    public List<LinhaDePesquisa> getLinhasComProjetos() {
        return getHibernateTemplate().find(
                "SELECT DISTINCT l FROM LinhaDePesquisa l "
                + "LEFT JOIN FETCH l.professores pf "
                + "LEFT JOIN FETCH pf.projetos pj "
                + "");
    }
}
```

OrientacaoDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.OrientacaoDAO;
import br.ufsc.inf.ppgcc.pojo.Orientacao;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.springframework.dao.support.DataAccessUtils;
import org.springframework.stereotype.Repository;

@Repository
public class OrientacaoDAOImpl extends CustomHibernateDaoSupport implements
OrientacaoDAO {

    public void save(Orientacao orientacao) {
        getHibernateTemplate().saveOrUpdate(orientacao);
    }

    public void delete(Orientacao orientacao) {
        getHibernateTemplate().delete(orientacao);
    }

    public List<Orientacao> getByProfessor(Professor professor) {
        return getHibernateTemplate().find
                ("SELECT DISTINCT o FROM Orientacao o "
                + "LEFT JOIN FETCH o.aluno a "
                + "LEFT JOIN FETCH a.defesa d "
                + "LEFT JOIN FETCH d.bancas "
                + "LEFT JOIN FETCH o.professor "
                + "WHERE o.professor = ? "
```

```
            + "AND o.concluida = false"
            , professor);
    }

    public int getNumOrientacoesEmAndamentoByProfessor(Professor professor) {
        return DataAccessUtils.intResult(getHibernateTemplate().find(
            "SELECT COUNT(o) FROM Orientacao o "
            + "WHERE o.professor = ? "
            + "AND o.concluida = false",
            professor));
    }
}
```

ProfessorDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.ProfessorDAO;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.FetchMode;
import org.hibernate.LockMode;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.MatchMode;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

@Repository
public class ProfessorDAOImpl extends CustomHibernateDaoSupport implements
ProfessorDAO, java.io.Serializable {

    public void save(Professor professor) {
        if (professor.getId() == null) {
            getHibernateTemplate().save(professor);
        } else {
            getHibernateTemplate().update(professor);
        }
    }

    public void delete(Professor professor) {
        getHibernateTemplate().bulkUpdate("delete from Professor where id = ?",
professor.getId());
        if (professor.getUsuario() != null) {
            getHibernateTemplate().delete(professor.getUsuario());
        }
    }

    public Professor getById(Integer id) {
        return getHibernateTemplate().get(Professor.class, id);
    }

    public List<Professor> getAll() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Professor.class)
                .addOrder(Order.asc("nome")));
```

```
    }

    public List<Professor> getAllFetch() {
        return getHibernateTemplate()
                .find("SELECT DISTINCT p FROM Professor p "
                + "LEFT JOIN FETCH p.linhasDePesquisa "
                + "LEFT JOIN FETCH p.projetos "
                + "ORDER BY p.nome");
    }

    public List<Professor> getByNome(String nome) {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Professor.class)
                .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY)
                .setFetchMode("linhasDePesquisa", FetchMode.JOIN)
                .setFetchMode("grupo", FetchMode.JOIN)
                .add(Restrictions.like("nome", nome, MatchMode.ANYWHERE)
                .ignoreCase())
                .addOrder(Order.asc("nome")));
    }

    public List<Professor> getSemUsuario() {
        return getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Professor.class)
                .setFetchMode("coordena", FetchMode.JOIN)
                .add(Restrictions.isNull("usuario"))
                .addOrder(Order.asc("nome")));
    }
}
```

ProjetoDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.ProjetoDAO;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Projeto;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.springframework.stereotype.Repository;

@Repository
public class ProjetoDAOImpl extends CustomHibernateDaoSupport implements ProjetoDAO {

    public void save(Projeto projeto) {
        getHibernateTemplate().saveOrUpdate(projeto);
    }

    public void delete(Projeto projeto) {
        getHibernateTemplate().delete(projeto);
    }

    public List<Projeto> getByProfessor(Professor professor) {
        return getHibernateTemplate().find("SELECT p FROM Projeto p "
                + "LEFT JOIN FETCH p.professor pf "
                + "WHERE pf = ?", professor);
    }
```

```
    public List<Projeto> getAll(Professor professor) {
        return getHibernateTemplate().find("SELECT p FROM Projeto p "
                + "LEFT JOIN FETCH p.professor");
    }

    public List<Projeto> getByLinhaDePesquisa(LinhaDePesquisa linhaDePesquisa) {
        return getHibernateTemplate().find("SELECT pj FROM Projeto pj "
                + "LEFT JOIN FETCH pj.professor pf "
                + "LEFT JOIN FETCH pf.linhasDePesquisa l "
                + "WHERE l = ? "
                + "ORDER BY pj.titulo ASC", linhaDePesquisa);
    }
}
```

UsuarioDAOImpl

```
package br.ufsc.inf.ppgcc.dao.impl;

import br.ufsc.inf.ppgcc.dao.UsuarioDAO;
import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Usuario;
import br.ufsc.inf.ppgcc.util.CustomHibernateDaoSupport;
import java.util.List;
import org.hibernate.FetchMode;
import org.hibernate.criterion.CriteriaSpecification;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.MatchMode;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.dao.support.DataAccessUtils;
import org.springframework.stereotype.Repository;

@Repository
public class UsuarioDAOImpl extends CustomHibernateDaoSupport implements UsuarioDAO {

    public void save(Usuario usuario) {
        if (usuario.getId() == null) {
            getHibernateTemplate().save(usuario);
        } else {
            getHibernateTemplate().update(usuario);
        }
    }

    public void delete(Usuario usuario) {
        getHibernateTemplate().delete(usuario);
    }

    public List<Usuario> getByPesquisa(String login, String acesso, int associado) {
        DetachedCriteria dc = DetachedCriteria
                .forClass(Usuario.class)
                .add(Restrictions.like("login", login, MatchMode.ANYWHERE).ignoreCase());

        if (!acesso.equals("Todos")) {
            dc.add(Restrictions.eq("acesso", acesso));
        }

        switch (associado) {
            case 0:
```

```
                break;
            case 1:
                dc.createCriteria("aluno",
CriteriaSpecification.LEFT_JOIN).add(Restrictions.isNull("usuario"));
                dc.createCriteria("professor",
CriteriaSpecification.LEFT_JOIN).add(Restrictions.isNull("usuario"));
                break;
            case 2:
                dc.createAlias("aluno", "a").add(Restrictions.isNotNull("a.usuario"));
                break;
            case 3:
                dc.createAlias("professor", "p").add(Restrictions.isNotNull("p.usuario"));
                break;
        }

        dc.addOrder(Order.asc("login"));

        return getHibernateTemplate().findByCriteria(dc);
    }

    public Usuario getByLogin(String login) {
        return (Usuario)
DataAccessUtils.uniqueResult(getHibernateTemplate().findByCriteria(DetachedCriteria
                .forClass(Usuario.class)
                .setFetchMode("professor.grupo", FetchMode.JOIN)
                .setFetchMode("professor.linhasDePesquisa", FetchMode.JOIN)
                .setFetchMode("professor.coordena", FetchMode.JOIN)
                .setFetchMode("professor.bancas", FetchMode.JOIN)
                .setFetchMode("professor.projetos", FetchMode.JOIN)
                .setFetchMode("professor.orientacoes", FetchMode.JOIN)
                .setFetchMode("professor.orientacoes.aluno", FetchMode.JOIN)
                .setFetchMode("professor.orientacoes.aluno.defesa", FetchMode.JOIN)
                .setFetchMode("professor.orientacoes.aluno.defesa.bancas", FetchMode.JOIN)
                .setFetchMode("aluno.orientacao.professor", FetchMode.JOIN)
                .setFetchMode("aluno.defesa.bancas", FetchMode.JOIN)
                .add(Restrictions.eq("login", login))));
    }

}
```

ComiteBancaBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Banca;
import br.ufsc.inf.ppgcc.service.BancaService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("comiteBancaBean")
public class ComiteBancaBean implements java.io.Serializable {
```

```
    @Autowired
    private BancaService bancaService;
    private Banca banca;

    public void aceitarProfessor() {
        try {
            banca.setStatus(Banca.ACEITO_COMITE);
            bancaService.update(banca);
            Utils.addMessage("Professor aceito na banca com sucesso.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao aceitar professor na banca." + e);
        }
    }

    public void recusarProfessor() {
        try {
            banca.setStatus(Banca.RECUSADO_COMITE);
            bancaService.update(banca);
            Utils.addMessage("Professor recusado da banca com sucesso.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao recusar professor da banca." + e);
        }
    }

    public List<Banca> getProfessoresComPendencia() {
        try {
            return bancaService.getBancasByStatus(Banca.PENDENTE);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores." + e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public List<Banca> getProfessoresAceitos() {
        try {
            return bancaService.getBancasByStatus(Banca.ACEITO_COMITE);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores." + e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public List<Banca> getProfessoresRecusados() {
        try {
            return bancaService.getBancasByStatus(Banca.RECUSADO_COMITE);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores." + e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public Banca getBanca() {
        return banca;
    }

    public void setBanca(Banca banca) {
        this.banca = banca;
    }
}
```

ConfiguracoesBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.service.AcessosService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("configuracoesBean")
public class ConfiguracoesBean {

    @Autowired
    private ProfessorService professorService;
    @Autowired
    private AcessosService acessosService;
    private List<Professor> professores;
    private Professor professor;

    public ConfiguracoesBean() {
    }

    public void salvar() {
        try {
            acessosService.alterarPresidenteComiteBanca(professor.getUsuario());
            Utils.addMessage("Presidente alterado com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar presidente." + ex.toString());
        }
    }

    public Professor getProfessor() {
        if (professores == null) {
            try {
                professores = professorService.getAll();
            } catch (Exception ex) {
                Utils.addExceptionMessage("Erro ao carregar professores.");
                professores = new ArrayList<Professor>(0);
            }
        }
        return professor;
    }

    public void setProfessor(Professor professor) {
        this.professor = professor;
    }

    public List<Professor> getProfessores() {
```

```
        return professores;
    }

    public void setProfessores(List<Professor> professores) {
        this.professores = professores;
    }

}
```

DefesaBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.mail.MailUtil;
import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.Orientacao;
import br.ufsc.inf.ppgcc.pojo.OrientacaoId;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.service.AlunoService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.service.UsuarioService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("session")
@Qualifier("defesaBean")
public class DefesaBean implements java.io.Serializable {

    @Autowired
    private AlunoService alunoService;
    @Autowired
    private  ProfessorService professorService;
    private Aluno aluno;

    @Autowired
    public DefesaBean(LoginBean loginBean) {
        aluno = loginBean.getUsuario().getAluno();

        if (aluno.getOrientacao() == null) {
            aluno.setOrientacao(new Orientacao());
        }
    }

    public boolean isTemOrientador() {
        return aluno.getOrientacao().getProfessor() != null;
    }

    public boolean isConviteEnviado() {
        return this.isTemOrientador() && aluno.getOrientacao().getConvite() == Boolean.TRUE;
    }

    public boolean isConviteRecusado() {
```

```
        return this.isTemOrientador() && aluno.getOrientacao().getAceito() == Boolean.FALSE;
    }

    public boolean isConviteAceito() {
        return this.isTemOrientador() && aluno.getOrientacao().getAceito() == Boolean.TRUE;
    }

    /*
     * Getters and Setters
     *
     */
    public Aluno getAluno() {
        return aluno;
    }

    public List<Professor> getProfessores() {
        try {
            return professorService.getAll();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar Professores." + ex.toString());
            return new ArrayList<Professor>(0);
        }
    }
}
```

DefesasBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Defesa;
import br.ufsc.inf.ppgcc.service.DefesaService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("defesasBean")
public class DefesasBean {

    @Autowired
    private DefesaService defesaService;

    public List<Defesa> getDefesas() {
        try {
            return defesaService.getByMostrar();
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao carregar defesas.");
            return new ArrayList<Defesa>(0);
        }
    }
}
```

GerenciaAlunoBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.service.AlunoService;
import java.util.List;
import javax.faces.bean.ManagedBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("gerenciaAlunoBean")
public class GerenciaAlunoBean implements java.io.Serializable {

    @Autowired
    private AlunoService alunoService;
    private String nome;
    private List<Aluno> alunos;

    public void pesquisar() {
        try {
            alunos = alunoService.getByNome(nome);
        } catch (Exception ex) {
        }
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public List<Aluno> getAlunos() {
        return alunos;
    }
}
```

GerenciaDefesasBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Defesa;
import br.ufsc.inf.ppgcc.pojo.datamodel.DefesaDataModel;
import br.ufsc.inf.ppgcc.service.DefesaService;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.List;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;
```

```
@Controller
@Scope("view")
@Qualifier("gerenciaDefesasBean")
public class GerenciaDefesasBean {

    @Autowired
    private NewEditDeleteSaveCancel controleBotoes;
    private DefesaService defesaService;
    private List<Defesa> defesas;
    private Defesa selectedDefesa;
    private Defesa defesaTemp;

    @Autowired
    public GerenciaDefesasBean(DefesaService defesaService) {
        try {
            defesas = defesaService.getAll();
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao carregar defesas.");
        }
        this.defesaService = defesaService;
    }

    public void onSelect(SelectEvent event) {
        defesaTemp = this.getSelectedDefesa().clone();
        controleBotoes.controlarOnSelect();
    }

    public void onUnselect(UnselectEvent event) {
        defesaTemp = new Defesa();
        controleBotoes.controlarOnUnselect();
    }

    public void botaoEditar() {
        controleBotoes.controlarBotaoEditar();
    }

    public void botaoSalvar() {
        try {
            defesaService.save(defesaTemp);

            if (this.isModoEditar()) {
                this.getSelectedDefesa().copy(defesaTemp);
            } else if (this.isModoNovo()) {
                defesas.add(defesaTemp);
            }

            controleBotoes.controlarBotaoSalvar();
            Utils.addMessage("Defesa salvo com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar defesa." + ex.toString());
        }
    }

    public void botaoCancelar() {
        defesaTemp = this.getSelectedDefesa().clone();
        if (selectedDefesa == null) {
            controleBotoes.controlarOnUnselect();
        } else {
            controleBotoes.controlarBotaoCancelar();
```

```
    }
}

public boolean isDesabilitarBotaoNovo() {
    return controleBotoes.isDesabilitarBotaoNovo();
}

public boolean isDesabilitarBotaoEditar() {
    return controleBotoes.isDesabilitarBotaoEditar();
}

public boolean isDesabilitarBotaoExcluir() {
    return controleBotoes.isDesabilitarBotaoExcluir();
}

public boolean isDesabilitarBotaoSalvar() {
    return controleBotoes.isDesabilitarBotaoSalvar();
}

public boolean isDesabilitarBotaoCancelar() {
    return controleBotoes.isDesabilitarBotaoCancelar();
}

public boolean isModoInicio() {
    return controleBotoes.isModoInicio();
}

public boolean isModoNovo() {
    return controleBotoes.isModoNovo();
}

public boolean isModoEditar() {
    return controleBotoes.isModoEditar();
}

/*
 * Getters and Setters
 *
 */
public NewEditDeleteSaveCancel getControleBotoes() {
    return controleBotoes;
}

public DefesaDataModel getDefesas() {
    return new DefesaDataModel(defesas);
}

public Defesa getSelectedDefesa() {
    return selectedDefesa != null ? selectedDefesa : new Defesa();
}

public void setSelectedDefesa(Defesa selectedDefesa) {
    this.selectedDefesa = selectedDefesa;
}

public Defesa getDefesaTemp() {
    return defesaTemp;
}
```

```
    public void setDefesaTemp(Defesa defesaTemp) {
        this.defesaTemp = defesaTemp;
    }
}
```

GerenciaGrupoBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Grupo;
import br.ufsc.inf.ppgcc.pojo.datamodel.GrupoDataModel;
import br.ufsc.inf.ppgcc.service.GrupoService;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.List;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("gerenciaGrupoBean")
public class GerenciaGrupoBean implements java.io.Serializable {

    private GrupoService grupoService;
    @Autowired
    private NewEditDeleteSaveCancel controleBotoes;
    private List<Grupo> grupos;
    private Grupo selectedGrupo;
    private Grupo grupoTemp;

    @Autowired
    public GerenciaGrupoBean(GrupoService grupoService) {
        try {
            grupos = grupoService.getAll();
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao carregar grupos.");
        }
        this.grupoService = grupoService;
    }

    public void onSelect(SelectEvent event) {
        grupoTemp = this.getSelectedGrupo().clone();
        controleBotoes.controlarOnSelect();
    }

    public void onUnselect(UnselectEvent event) {
        grupoTemp = new Grupo();
        controleBotoes.controlarOnUnselect();
    }

    public void botaoNovo() {
        grupoTemp = new Grupo();
        controleBotoes.controlarBotaoNovo();
    }
```

```
    public void botaoEditar() {
        controleBotoes.controlarBotaoEditar();
    }

    public void botaoExcluir() {
        try {
            grupoService.delete(selectedGrupo);
            grupos.remove(this.getSelectedGrupo());
            selectedGrupo = null;
            grupoTemp = new Grupo();
            controleBotoes.controlarBotaoExcluir();
            Utils.addMessage("Grupo excluído com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao excluir grupo." + ex.toString());
        }
    }

    public void botaoSalvar() {
        try {
            grupoService.save(grupoTemp);

            if (this.isModoEditar()) {
                this.getSelectedGrupo().copy(grupoTemp);
            } else if (this.isModoNovo()) {
                grupos.add(grupoTemp);
            }

            controleBotoes.controlarBotaoSalvar();
            Utils.addMessage("Grupo salvo com sucesso.");
        } catch (DataIntegrityViolationException ex) {
            Utils.addSpecificExceptionMessage("pitGrupoNome", "Já existe um grupo com este
nome.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar grupo." + ex.toString());
        }
    }

    public void botaoCancelar() {
        grupoTemp = this.getSelectedGrupo().clone();
        if (selectedGrupo == null) {
            controleBotoes.controlarOnUnselect();
        } else {
            controleBotoes.controlarBotaoCancelar();
        }
    }

    public boolean isDesabilitarBotaoNovo() {
        return controleBotoes.isDesabilitarBotaoNovo();
    }

    public boolean isDesabilitarBotaoEditar() {
        return controleBotoes.isDesabilitarBotaoEditar();
    }

    public boolean isDesabilitarBotaoExcluir() {
        return controleBotoes.isDesabilitarBotaoExcluir();
    }
```

```
    public boolean isDesabilitarBotaoSalvar() {
        return controleBotoes.isDesabilitarBotaoSalvar();
    }

    public boolean isDesabilitarBotaoCancelar() {
        return controleBotoes.isDesabilitarBotaoCancelar();
    }

    public boolean isModoInicio() {
        return controleBotoes.isModoInicio();
    }

    public boolean isModoNovo() {
        return controleBotoes.isModoNovo();
    }

    public boolean isModoEditar() {
        return controleBotoes.isModoEditar();
    }

    /*
     * Getters and Setters
     *
     */
    public NewEditDeleteSaveCancel getControleBotoes() {
        return controleBotoes;
    }

    public GrupoDataModel getGrupos() {
        return new GrupoDataModel(grupos);
    }

    public Grupo getSelectedGrupo() {
        return selectedGrupo != null ? selectedGrupo : new Grupo();
    }

    public void setSelectedGrupo(Grupo selectedGrupo) {
        this.selectedGrupo = selectedGrupo;
    }

    public Grupo getGrupoTemp() {
        return grupoTemp;
    }

    public void setGrupoTemp(Grupo grupoTemp) {
        this.grupoTemp = grupoTemp;
    }
}
```

GerenciaLinhaDePesquisaBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.datamodel.LinhaDePesquisaDataModel;
import br.ufsc.inf.ppgcc.service.LinhaDePesquisaService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
```

```
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.primefaces.model.DualListModel;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("gerenciaLinhaDePesquisaBean")
public class GerenciaLinhaDePesquisaBean implements java.io.Serializable {

    @Autowired
    private LinhaDePesquisaService linhaDePesquisaService;
    @Autowired
    private  ProfessorService professorService;
    @Autowired
    private NewEditDeleteSaveCancel controleBotoes;
    private String pesquisaTitulo;
    private List<LinhaDePesquisa> linhas = new ArrayList<LinhaDePesquisa>();
    private LinhaDePesquisa selectedLinha;
    private LinhaDePesquisa linhaTemp;
    private DualListModel<Professor> professores;

    public GerenciaLinhaDePesquisaBean() {
        linhaTemp = new LinhaDePesquisa();
    }

    public void pesquisar() {
        try {
            linhas = linhaDePesquisaService.getByTitulo(pesquisaTitulo);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao executar pesquisa." + ex.toString());
        }
    }

    public void onSelect(SelectEvent event) {
        linhaTemp = this.getSelectedLinha().clone();
        controleBotoes.controlarOnSelect();
    }

    public void onUnselect(UnselectEvent event) {
        linhaTemp = new LinhaDePesquisa();
        controleBotoes.controlarOnUnselect();
    }

    public void botaoNovo() {
        linhaTemp = new LinhaDePesquisa();
        controleBotoes.controlarBotaoNovo();
    }

    public void botaoEditar() {
        controleBotoes.controlarBotaoEditar();
    }
```

```
public void botaoExcluir() {
    try {
        linhaDePesquisaService.delete(selectedLinha);
        linhas.remove(this.getSelectedLinha());
        selectedLinha = null;
        linhaTemp = new LinhaDePesquisa();
        controleBotoes.controlarBotaoExcluir();
        Utils.addMessage("Linha de Pesquisa excluída com sucesso.");
    } catch (Exception ex) {
        Utils.addExceptionMessage("Erro ao excluir Linha de Pesquisa." + ex.toString());
    }
}

public void botaoSalvar() {
    try {
        linhaTemp.setProfessores(new HashSet<Professor>(professores.getTarget()));
        linhaDePesquisaService.save(linhaTemp);

        if (this.isModoEditar()) {
            this.getSelectedLinha().copy(linhaTemp);
        }

        controleBotoes.controlarBotaoSalvar();
        Utils.addMessage("Linha de Pesquisa salva com sucesso.");
    } catch (Exception ex) {
        Utils.addExceptionMessage("Erro ao salvar Linha de Pesquisa." + ex.toString());
    }
}

public void botaoCancelar() {
    linhaTemp = this.getSelectedLinha().clone();
    if (selectedLinha == null) {
        controleBotoes.controlarOnUnselect();
    } else {
        controleBotoes.controlarBotaoCancelar();
    }
}

public boolean isDesabilitarBotaoNovo() {
    return controleBotoes.isDesabilitarBotaoNovo();
}

public boolean isDesabilitarBotaoEditar() {
    return controleBotoes.isDesabilitarBotaoEditar();
}

public boolean isDesabilitarBotaoExcluir() {
    return controleBotoes.isDesabilitarBotaoExcluir();
}

public boolean isDesabilitarBotaoSalvar() {
    return controleBotoes.isDesabilitarBotaoSalvar();
}

public boolean isDesabilitarBotaoCancelar() {
    return controleBotoes.isDesabilitarBotaoCancelar();
}
```

```
public boolean isModoInicio() {
    return controleBotoes.isModoInicio();
}

public boolean isModoNovo() {
    return controleBotoes.isModoNovo();
}

public boolean isModoEditar() {
    return controleBotoes.isModoEditar();
}

/*
 * Getters and Setters
 *
 */
public NewEditDeleteSaveCancel getControleBotoes() {
    return controleBotoes;
}

public String getPesquisaTitulo() {
    return pesquisaTitulo;
}

public void setPesquisaTitulo(String pesquisaTitulo) {
    this.pesquisaTitulo = pesquisaTitulo;
}

public LinhaDePesquisaDataModel getLinhas() {
    return new LinhaDePesquisaDataModel(linhas);
}

public LinhaDePesquisa getSelectedLinha() {
    return selectedLinha != null ? selectedLinha : new LinhaDePesquisa();
}

public void setSelectedLinha(LinhaDePesquisa selectedLinha) {
    this.selectedLinha = selectedLinha;
}

public LinhaDePesquisa getLinhaTemp() {
    return linhaTemp;
}

public void setLinhaTemp(LinhaDePesquisa linhaTemp) {
    this.linhaTemp = linhaTemp;
}

public List<Professor> getProfessoresSelect() {
    try {
        return professorService.getAll();
    } catch (Exception ex) {
        Utils.addExceptionMessage("Erro ao carregar Professores." + ex.toString());
        return new ArrayList<Professor>(0);
    }
}

public DualListModel<Professor> getProfessores() {
    try {
```

```
            List<Professor> professoresDisponiveis = professorService.getAll();
            List<Professor> professoresLinha = linhaTemp.getProfessoresList();
            professoresDisponiveis.removeAll(professoresLinha);
            professores = new DualListModel<Professor>(professoresDisponiveis,
professoresLinha);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar Professores." + ex.toString());
        }

        return professores;
    }

    public void setProfessores(DualListModel<Professor> professores) {
        this.professores = professores;
    }
}
```

GerenciaProfessorBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Grupo;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.datamodel.ProfessorDataModel;
import br.ufsc.inf.ppgcc.service.GrupoService;
import br.ufsc.inf.ppgcc.service.LinhaDePesquisaService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import javax.faces.bean.ManagedBean;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.primefaces.model.DualListModel;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("gerenciaProfessorBean")
public class GerenciaProfessorBean implements java.io.Serializable {

    @Autowired
    private ProfessorService professorService;
    @Autowired
    private LinhaDePesquisaService linhaDePesquisaService;
    @Autowired
    private GrupoService grupoService;
    @Autowired
    private NewEditDeleteSaveCancel controleBotoes;
    private String pesquisaNome;
    private List<Professor> professores = new ArrayList<Professor>();
    private Professor selectedProfessor;
    private Professor professorTemp;
```

```
    private List<LinhaDePesquisa> linhasTodas;
    private DualListModel<LinhaDePesquisa> linhas;
    private List<Grupo> grupos;

    public void pesquisar() {
        try {
            professores = professorService.getByNome(pesquisaNome);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao executar pesquisa." + ex.toString());
        }
    }

    public void onSelect(SelectEvent event) {
        professorTemp = this.getSelectedProfessor().clone();
        controleBotoes.controlarOnSelect();
    }

    public void onUnselect(UnselectEvent event) {
        professorTemp = new Professor();
        controleBotoes.controlarOnUnselect();
    }

    public void botaoNovo() {
        this.carregarObjetos();
        professorTemp = new Professor();
        controleBotoes.controlarBotaoNovo();
    }

    public void botaoEditar() {
        this.carregarObjetos();
        controleBotoes.controlarBotaoEditar();
    }

    public void botaoExcluir() {
        try {
            professorService.delete(selectedProfessor);
            professores.remove(this.getSelectedProfessor());
            selectedProfessor = null;
            professorTemp = new Professor();
            controleBotoes.controlarBotaoExcluir();
            Utils.addMessage("Professor excluído com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao excluir professor." + ex.toString());
        }
    }

    public void botaoSalvar() {
        try {
            professorTemp.setLinhasDePesquisa(new
HashSet<LinhaDePesquisa>(linhas.getTarget()));
            professorService.save(professorTemp);

            if (this.isModoEditar()) {
                this.getSelectedProfessor().copy(professorTemp);
            }

            controleBotoes.controlarBotaoSalvar();
            Utils.addMessage("Professor salvo com sucesso.");
        } catch (Exception ex) {
```

```
            Utils.addExceptionMessage("Erro ao salvar professor." + ex.toString());
        }
    }

    public void botaoCancelar() {
        professorTemp = this.getSelectedProfessor().clone();
        if (selectedProfessor == null) {
            controleBotoes.controlarOnUnselect();
        } else {
            controleBotoes.controlarBotaoCancelar();
        }
    }

    public boolean isDesabilitarBotaoNovo() {
        return controleBotoes.isDesabilitarBotaoNovo();
    }

    public boolean isDesabilitarBotaoEditar() {
        return controleBotoes.isDesabilitarBotaoEditar();
    }

    public boolean isDesabilitarBotaoExcluir() {
        return controleBotoes.isDesabilitarBotaoExcluir();
    }

    public boolean isDesabilitarBotaoSalvar() {
        return controleBotoes.isDesabilitarBotaoSalvar();
    }

    public boolean isDesabilitarBotaoCancelar() {
        return controleBotoes.isDesabilitarBotaoCancelar();
    }

    public boolean isModoInicio() {
        return controleBotoes.isModoInicio();
    }

    public boolean isModoNovo() {
        return controleBotoes.isModoNovo();
    }

    public boolean isModoEditar() {
        return controleBotoes.isModoEditar();
    }

    private void carregarObjetos() {
        this.carregarLinhas();
        this.carregarGrupos();
    }

    private void carregarLinhas() {
        if (linhasTodas == null) {
            try {
                linhasTodas = linhaDePesquisaService.getAll();
            } catch (Exception e) {
                Utils.addExceptionMessage("Erro ao carregar Linhas de Pesquisa." + e.toString());
            }
        }
    }
```

```
    private void carregarGrupos() {
        if (grupos == null) {
            try {
                grupos = grupoService.getAll();
            } catch (Exception e) {
                Utils.addExceptionMessage("Erro ao carregar grupos." + e.toString());
            }
        }
    }

    /*
     * Getters and Setters
     *
     */
    public NewEditDeleteSaveCancel getControleBotoes() {
        return controleBotoes;
    }

    public String getPesquisaNome() {
        return pesquisaNome;
    }

    public void setPesquisaNome(String pesquisaNome) {
        this.pesquisaNome = pesquisaNome;
    }

    public ProfessorDataModel getProfessores() {
        return new ProfessorDataModel(professores);
    }

    public Professor getSelectedProfessor() {
        return selectedProfessor != null ? selectedProfessor : new Professor();
    }

    public void setSelectedProfessor(Professor selectedProfessor) {
        this.selectedProfessor = selectedProfessor;
    }

    public Professor getProfessorTemp() {
        return professorTemp;
    }

    public void setProfessorTemp(Professor professorTemp) {
        this.professorTemp = professorTemp;
    }

    public DualListModel<LinhaDePesquisa> getLinhas() {
        try {
            List<LinhaDePesquisa> linhasDisponiveis = new
ArrayList<LinhaDePesquisa>(linhasTodas);
            List<LinhaDePesquisa> linhasProfessor = new
ArrayList<LinhaDePesquisa>(professorTemp.getLinhasDePesquisa());
            linhasDisponiveis.removeAll(linhasProfessor);
            linhas = new DualListModel<LinhaDePesquisa>(linhasDisponiveis, linhasProfessor);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar Linhas de Pesquisa." + ex.toString());
        }
        return linhas;
```

```
    }

    public void setLinhas(DualListModel<LinhaDePesquisa> linhas) {
        this.linhas = linhas;
    }

    public List<Grupo> getGrupos() {
        return grupos;
    }
}
```

GerenciaUsuarioBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.service.UsuarioService;
import br.ufsc.inf.ppgcc.mail.MailUtil;
import br.ufsc.inf.ppgcc.pojo.Acessos;
import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Usuario;
import br.ufsc.inf.ppgcc.pojo.datamodel.UsuarioDataModel;
import br.ufsc.inf.ppgcc.security.Acesso;
import br.ufsc.inf.ppgcc.service.AcessosService;
import br.ufsc.inf.ppgcc.service.AlunoService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.model.SelectItem;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("gerenciaUsuarioBean")
public class GerenciaUsuarioBean implements java.io.Serializable {

    @Autowired
    private UsuarioService usuarioService;
    @Autowired
    private AlunoService alunoService;
    @Autowired
    private ProfessorService professorService;
    @Autowired
    private AcessosService acessosService;
    @Autowired
    private NewEditDeleteSaveCancel controleBotoes;
    private String pesquisaLogin;
    private String pesquisaAcesso;
    private int pesquisaAssociado;
    private List<SelectItem> pesquisaAcessos;
```

```
    private List<SelectItem> pesquisaAssociados;
    private List<Usuario> usuarios = new ArrayList<Usuario>();
    private Usuario selectedUsuario;
    private Usuario usuarioTemp;
    private List<SelectItem> acessos;
    private int associado;
    private List<Aluno> alunos;
    private List<Professor> professores;
    private String email;
    private Professor selectedProfessor;
    private String acesso;
    private UsuarioDataModel usuarioDataModel;

    public GerenciaUsuarioBean() {
        usuarioTemp = new Usuario();

        this.popularListPesquisaAcessos();
        this.popularListPesquisaAssociados();
        this.popularListAcessos();
    }

    public void pesquisar() {
        try {
            usuarios = usuarioService.getByPesquisa(pesquisaLogin, pesquisaAcesso,
pesquisaAssociado);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao executar pesquisa." + ex.toString());
        }
    }

    public void onSelect(SelectEvent event) {
        usuarioTemp = this.getSelectedUsuario().clone();
        controleBotoes.controlarOnSelect();
    }

    public void onUnselect(UnselectEvent event) {
        usuarioTemp = new Usuario();
        controleBotoes.controlarOnUnselect();
    }

    public void botaoNovo() {
        this.carregarObjetos();
        usuarioTemp = new Usuario();
        associado = 0;
        email = "";
        acesso = "";
        controleBotoes.controlarBotaoNovo();
    }

    public void botaoEditar() {
        acesso = ((Acessos) usuarioTemp.getAcessos().toArray()[0]).getAcesso();
        controleBotoes.controlarBotaoEditar();
    }

    public void botaoExcluir() {
        try {
            usuarioService.delete(selectedUsuario);
            usuarios.remove(this.getSelectedUsuario());
            selectedUsuario = null;
```

```
            usuarioTemp = new Usuario();
            controleBotoes.controlarBotaoExcluir();
            Utils.addMessage("Usuário excluído com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao excluir usuário." + ex.toString());
        }
    }

    public void botaoSalvar() throws Exception {
        try {
            Acessos acess;
            String senha = "";

            if (this.isModoNovo()) {
                senha = Utils.gerarSenha();
                usuarioTemp.setSenha(Utils.criptografarSenha(senha));
                if (!acesso.equals("")) {
                    acess = new Acessos();
                    acess.setAcesso(acesso);
                    acess.setUsuario(usuarioTemp);
                    usuarioTemp.getAcessos().add(acess);
                }
            } else {
                Acessos acessosTemp = new Acessos();
                acessosTemp.setUsuario(usuarioTemp);
                if (acesso.equals("")) {
                    acessosTemp.setAcesso(Acesso.ADMINISTRADOR.getAuthority());
                    if (usuarioTemp.getAcessos().contains(acessosTemp)) {
                        acessosService.deletar(acessosTemp);
                        usuarioTemp.getAcessos().remove(acessosTemp);
                    } else {
                        acessosTemp.setAcesso(Acesso.LEITURA.getAuthority());
                        if (usuarioTemp.getAcessos().contains(acessosTemp)) {
                            acessosService.deletar(acessosTemp);
                            usuarioTemp.getAcessos().remove(acessosTemp);
                        }
                    }
                } else if (acesso.equals(Acesso.ADMINISTRADOR.getAuthority())) {
                    acessosTemp.setAcesso(Acesso.LEITURA.getAuthority());
                    if (usuarioTemp.getAcessos().contains(acessosTemp)) {
                        acessosService.deletar(acessosTemp);
                        usuarioTemp.getAcessos().remove(acessosTemp);
                    }

                    acessosTemp.setAcesso(Acesso.ADMINISTRADOR.getAuthority());
                    if (!usuarioTemp.getAcessos().contains(acessosTemp)) {
                        acess = new Acessos();
                        acess.setAcesso(acesso);
                        acess.setUsuario(usuarioTemp);
                        usuarioTemp.getAcessos().add(acess);
                    }
                } else if (acesso.equals(Acesso.LEITURA.getAuthority())) {
                    acessosTemp.setAcesso(Acesso.ADMINISTRADOR.getAuthority());
                    if (usuarioTemp.getAcessos().contains(acessosTemp)) {
                        acessosService.deletar(acessosTemp);
                        usuarioTemp.getAcessos().remove(acessosTemp);
                    }

                    acessosTemp.setAcesso(Acesso.LEITURA.getAuthority());
```

```
                if (!usuarioTemp.getAcessos().contains(acessosTemp)) {
                    acess = new Acessos();
                    acess.setAcesso(acesso);
                    acess.setUsuario(usuarioTemp);
                    usuarioTemp.getAcessos().add(acess);
                }
            }
        }

        if (!this.isAssociadoAluno()) {
            usuarioTemp.setAluno(null);
        } else {
            usuarioTemp.getAluno().setUsuario(usuarioTemp);
            if (this.isModoNovo()) {
                acess = new Acessos();
                acess.setAcesso(Acesso.ALUNO.getAuthority());
                acess.setUsuario(usuarioTemp);
                usuarioTemp.getAcessos().add(acess);
            }
        }

        if (!this.isAssociadoProfessor()) {
            usuarioTemp.setProfessor(null);
        } else {
            usuarioTemp.getProfessor().setUsuario(usuarioTemp);
            if (this.isModoNovo()) {
                acess = new Acessos();
                acess.setAcesso(Acesso.PROFESSOR.getAuthority());
                acess.setUsuario(usuarioTemp);
                usuarioTemp.getAcessos().add(acess);
            }
        }

        System.out.println(usuarioTemp.getAcessos());
        usuarioService.save(usuarioTemp);

        if (this.isModoNovo()) {
            MailUtil.getInstance().enviarEmailNovoUsuario(usuarioTemp, email, senha);
        }

        if (this.isModoEditar()) {
            this.getSelectedUsuario().copy(usuarioTemp);
        }

        controleBotoes.controlarBotaoSalvar();
        if (this.isModoNovo()) {
            Utils.addMessage("Usuário salvo com sucesso. Um e-mail com o login e a senha foi
enviado para " + email + "." + senha);
        } else {
            Utils.addMessage("Usuário salvo com sucesso.");
        }
      } catch (DataIntegrityViolationException ex) {
          Utils.addWarningMessage("O login '" + usuarioTemp.getLogin() + "' já existe." +
ex.getMessage());
      } catch (Exception ex) {
          Utils.addExceptionMessage("Erro ao salvar usuário." + ex.toString());
      }
   }
```

```
    public void botaoCancelar() {
        usuarioTemp = this.getSelectedUsuario().clone();
        if (selectedUsuario == null) {
            controleBotoes.controlarOnUnselect();
        } else {
            controleBotoes.controlarBotaoCancelar();
        }
    }

    public void carregarEmailAluno() {
        if (usuarioTemp.getAluno() != null) {
            email = usuarioTemp.getAluno().getEmail();
        }
    }

    public void carregarEmailProfessor() {
        if (usuarioTemp.getProfessor() != null) {
            email = usuarioTemp.getProfessor().getEmail();
            System.out.println(email);
        }
    }

    public boolean isAssociadoAluno() {
        return controleBotoes.isModoNovo() ? associado == 1 :
usuarioTemp.isAssociadoAAluno();
    }

    public boolean isAssociadoProfessor() {
        return controleBotoes.isModoNovo() ? associado == 2 :
usuarioTemp.isAssociadoAProfessor();
    }

    private boolean isUsuarioCoordenador() {
        return usuarioTemp.isAssociadoACoordenador();
    }

    public boolean isDesabilitarBotaoNovo() {
        return controleBotoes.isDesabilitarBotaoNovo();
    }

    public boolean isDesabilitarBotaoEditar() {
        return controleBotoes.isDesabilitarBotaoEditar();
    }

    public boolean isDesabilitarBotaoExcluir() {
        return controleBotoes.isDesabilitarBotaoExcluir();
    }

    public boolean isDesabilitarBotaoSalvar() {
        return controleBotoes.isDesabilitarBotaoSalvar();
    }

    public boolean isDesabilitarBotaoCancelar() {
        return controleBotoes.isDesabilitarBotaoCancelar();
    }

    public boolean isModoInicio() {
        return controleBotoes.isModoInicio();
    }
```

```
    public boolean isModoNovo() {
        return controleBotoes.isModoNovo();
    }

    public boolean isModoEditar() {
        return controleBotoes.isModoEditar();
    }

    public String getNomeAssociado() {
        return this.isAssociadoAluno() ? usuarioTemp.getAluno().getNome()
                : this.isAssociadoProfessor() ? usuarioTemp.getProfessor().getNome()
                : selectedUsuario != null ? "Sem associação"
                : "";
    }

    private void popularListPesquisaAcessos() {
        pesquisaAcessos = new ArrayList<SelectItem>(6);
        pesquisaAcessos.add(new SelectItem("Todos", "Todos"));
        pesquisaAcessos.add(new SelectItem(Acesso.ADMINISTRADOR.getAuthority(),
"Administrador"));
        pesquisaAcessos.add(new SelectItem(Acesso.LEITURA.getAuthority(), "Leitura"));
        pesquisaAcessos.add(new SelectItem(Acesso.COORDENADOR.getAuthority(),
"Coordenador"));
        pesquisaAcessos.add(new SelectItem(Acesso.PROFESSOR.getAuthority(), "Professor"));
        pesquisaAcessos.add(new SelectItem(Acesso.ALUNO.getAuthority(), "Aluno"));
    }

    private void popularListPesquisaAssociados() {
        pesquisaAssociados = new ArrayList<SelectItem>(4);
        pesquisaAssociados.add(new SelectItem(0, "Qualquer um"));
        pesquisaAssociados.add(new SelectItem(1, "Sem associação"));
        pesquisaAssociados.add(new SelectItem(2, "Aluno"));
        pesquisaAssociados.add(new SelectItem(3, "Professor"));
    }

    private void popularListAcessos() {
        SelectItem si = new SelectItem();
        si.setValue("");
        si.setLabel("Selecione um acesso");
        si.setNoSelectionOption(true);
        acessos = new ArrayList<SelectItem>(3);
        acessos.add(si);
        acessos.add(new SelectItem(Acesso.ADMINISTRADOR.getAuthority(), "Administrador"));
        acessos.add(new SelectItem(Acesso.LEITURA.getAuthority(), "Leitura"));
    }

    private void carregarObjetos() {
        this.carregarAlunos();
        this.carregarProfessores();
    }

    private void carregarAlunos() {
        if (alunos == null) {
            try {
                alunos = alunoService.getSemUsuario();
            } catch (Exception ex) {
                Utils.addExceptionMessage("Erro ao carregar alunos.");
                alunos = new ArrayList<Aluno>(0);
```

```
        }
    }
}

private void carregarProfessores() {
    if (professores == null) {
        try {
            professores = professorService.getSemUsuario();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar professores.");
            professores = new ArrayList<Professor>(0);
        }
    }
}

/*
 * Getters and Setters
 *
 */
public UsuarioDataModel getUsuarioDataModel() {
    usuarioDataModel = new UsuarioDataModel(usuarios);
    return usuarioDataModel;
}

public NewEditDeleteSaveCancel getControleBotoes() {
    return controleBotoes;
}

public String getPesquisaLogin() {
    return pesquisaLogin;
}

public void setPesquisaLogin(String pesquisaLogin) {
    this.pesquisaLogin = pesquisaLogin;
}

public String getPesquisaAcesso() {
    return pesquisaAcesso;
}

public void setPesquisaAcesso(String pesquisaAcesso) {
    this.pesquisaAcesso = pesquisaAcesso;
}

public int getPesquisaAssociado() {
    return pesquisaAssociado;
}

public void setPesquisaAssociado(int pesquisaAssociado) {
    this.pesquisaAssociado = pesquisaAssociado;
}

public Usuario getSelectedUsuario() {
    return selectedUsuario != null ? selectedUsuario : new Usuario();
}

public void setSelectedUsuario(Usuario selectedUsuario) {
    this.selectedUsuario = selectedUsuario;
}
```

```
public Usuario getUsuarioTemp() {
    return usuarioTemp;
}

public void setUsuarioTemp(Usuario usuarioTemp) {
    this.usuarioTemp = usuarioTemp;
}

public List<Usuario> getUsuarios() {
    return usuarios;
}

public void setUsuarios(List<Usuario> usuarios) {
    this.usuarios = usuarios;
}

public List<SelectItem> getAcessos() {
    return acessos;
}

public List<SelectItem> getPesquisaAcessos() {
    return pesquisaAcessos;
}

public List<SelectItem> getPesquisaAssociados() {
    return pesquisaAssociados;
}

public int getAssociado() {
    return associado;
}

public void setAssociado(int associado) {
    this.associado = associado;
}

public List<Professor> getProfessores() {
    return professores;
}

public void setProfessores(List<Professor> professores) {
    this.professores = professores;
}

public List<Aluno> getAlunos() {
    return alunos;
}

public void setAlunos(List<Aluno> alunos) {
    this.alunos = alunos;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
```

```
    }

    public Professor getSelectedProfessor() {
        return selectedProfessor;
    }

    public void setSelectedProfessor(Professor selectedProfessor) {
        this.selectedProfessor = selectedProfessor;
    }

    public String getAcesso() {
        return acesso;
    }

    public void setAcesso(String acesso) {
        this.acesso = acesso;
    }
}
```

InscricaoBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.ConhecimentoLinguas;
import br.ufsc.inf.ppgcc.pojo.Curriculo;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.datamodel.LinguaDataModel;
import br.ufsc.inf.ppgcc.recommendation.RecomendacaoPessoa;
import br.ufsc.inf.ppgcc.recommendation.SistemaDeRecomendacao;
import br.ufsc.inf.ppgcc.service.AlunoService;
import br.ufsc.inf.ppgcc.service.LinhaDePesquisaService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import org.primefaces.event.FlowEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("inscricaoBean")
public class InscricaoBean implements java.io.Serializable {

    @Autowired
    private AlunoService alunoService;
    @Autowired
    private LinhaDePesquisaService linhaDePesquisaService;
    @Autowired
    private ProfessorService professorService;
```

```
    private Aluno aluno;
    private Curriculo curriculo;
    private Curriculo selectedCurriculo;
    private Curriculo curriculoTemp;
    private ConhecimentoLinguas conhecimentoLinguas;
    private ConhecimentoLinguas selectedConhecimentoLinguas;
    private List<LinhaDePesquisa> linhasDePesquisa;
    private List<Professor> professores;
    private List<Curriculo> curriculos;
    private List<ConhecimentoLinguas> linguas;
    private boolean inscricaoCompleta;
    private boolean confirmar;
    private static final Map<String,String> nivelConhecimentoLingua;
    List<RecomendacaoPessoa<Professor>> recomendacoes = new
ArrayList<RecomendacaoPessoa<Professor>>(0);
    private static int count = 0;

    static {
        nivelConhecimentoLingua = new LinkedHashMap<String, String>();
        nivelConhecimentoLingua.put("Excelente", "Excelente");
        nivelConhecimentoLingua.put("Bom", "Bom");
        nivelConhecimentoLingua.put("Regular", "Regular");
        nivelConhecimentoLingua.put("Pouco", "Pouco");
    }

    @Autowired
    public InscricaoBean(LinhaDePesquisaService linhaDePesquisaService) {
        aluno = new Aluno();

        curriculos = new ArrayList<Curriculo>();
        curriculo = new Curriculo();
        selectedCurriculo = new Curriculo();
        curriculoTemp = new Curriculo();

        conhecimentoLinguas = new ConhecimentoLinguas();
        linguas = new ArrayList<ConhecimentoLinguas>();

        aluno.setEstadoCivil("Selecione");
        aluno.setSexo("Selecione");

    }

    public void recomendarProfessores() {
        try {
            recomendacoes = SistemaDeRecomendacao.recomendarProfessores(aluno,
professorService.getAllFetch());
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao recomendar professores.");
        }
    }

    public void inscrever() {
        aluno.setCurriculos(new HashSet<Curriculo>(curriculos));
        aluno.setConhecimentosLinguas(new HashSet<ConhecimentoLinguas>(linguas));
        try {
            alunoService.save(aluno);
            inscricaoCompleta = true;
            Utils.addMessage("InscriÃ§Ã£o realizada com sucesso.");
        } catch (Exception ex) {
```

```
            Utils.addExceptionMessage("Erro ao realizar inscriÃ§Ã£o.");
        }
    }

    public void novoCurriculo() {
        curriculo = new Curriculo();
    }

    public void novaLingua() {
        conhecimentoLinguas = new ConhecimentoLinguas();
    }

    public void atualizarCurriculo() {
        selectedCurriculo.copy(curriculoTemp);
    }

    public void clonarCurriculo() {
        curriculoTemp = selectedCurriculo.clone();
    }

    public void handleLinhaChange() {
        try {
            professores = aluno.getDadosInscricao().getLinhaDePesquisa().getProfessoresList();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar Professores");
        }
    }

    /*
     *
     * Getters and Setters
     *
     */
    public List<RecomendacaoPessoa<Professor>> getRecomendacoes() {
        return recomendacoes;
    }

    public Aluno getAluno() {
        return aluno;
    }

    public void setAluno(Aluno aluno) {
        this.aluno = aluno;
    }

    public Curriculo getCurriculo() {
        return curriculo;
    }

    public void setCurriculo(Curriculo curriculo) {
        this.curriculo = curriculo;
    }

    public Curriculo getSelectedCurriculo() {
        return selectedCurriculo;
    }

    public void setSelectedCurriculo(Curriculo selectedCurriculo) {
        this.selectedCurriculo = selectedCurriculo;
```

```
}

public Curriculo getCurriculoTemp() {
    return curriculoTemp;
}

public void setCurriculoTemp(Curriculo curriculoTemp) {
    this.curriculoTemp = curriculoTemp;
}

public List<Curriculo> getCurriculos() {
    return curriculos;
}

public void setCurriculos(List<Curriculo> curriculos) {
    this.curriculos = curriculos;
}

public Map<String,String> getNivelConhecimentoLingua() {
    return nivelConhecimentoLingua;
}

public List<LinhaDePesquisa> getLinhasDePesquisa() {
    try {
        linhasDePesquisa = linhaDePesquisaService.getAll();
    } catch (Exception ex) {
        Utils.addExceptionMessage("Erro ao carregar Linhas de Pesquisa.");
    }
    return linhasDePesquisa;
}

public void setLinhasDePesquisa(List<LinhaDePesquisa> linhasDePesquisa) {
    this.linhasDePesquisa = linhasDePesquisa;
}

public List<Professor> getProfessores() {
    return professores;
}

public void setProfessores(List<Professor> professores) {
    this.professores = professores;
}

public LinguaDataModel getLinguasDataModel() {
    return new LinguaDataModel(linguas);
}

public List<ConhecimentoLinguas> getLinguas() {
    return linguas;
}

public void setLinguas(List<ConhecimentoLinguas> linguas) {
    this.linguas = linguas;
}

public ConhecimentoLinguas getConhecimentoLinguas() {
    return conhecimentoLinguas;
}
```

```
    public void setConhecimentoLinguas(ConhecimentoLinguas conhecimentoLinguas) {
        this.conhecimentoLinguas = conhecimentoLinguas;
    }

    public ConhecimentoLinguas getSelectedConhecimentoLinguas() {
        return selectedConhecimentoLinguas;
    }

    public void setSelectedConhecimentoLinguas(ConhecimentoLinguas
selectedConhecimentoLinguas) {
        this.selectedConhecimentoLinguas = selectedConhecimentoLinguas;
    }

    public boolean isInscricaoCompleta() {
        return inscricaoCompleta;
    }

    public boolean isConfirmar() {
        return confirmar;
    }
}
```

LinhasDePesquisaBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Projeto;
import br.ufsc.inf.ppgcc.service.LinhaDePesquisaService;
import br.ufsc.inf.ppgcc.service.ProjetoService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import org.primefaces.event.TabChangeEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("linhasDePesquisaBean")
public class LinhasDePesquisaBean {

    @Autowired
    private LinhaDePesquisaService linhaDePesquisaService;
    @Autowired
    private ProjetoService projetoService;
    private List<LinhaDePesquisa> linhasDePesquisa;
    private Professor selectedProfessor;
    private Projeto selectedProjeto;
    private List<Projeto> projetos;

    public List<LinhaDePesquisa> getLinhasDePesquisa() {
        if (linhasDePesquisa == null) {
            try {
                return linhasDePesquisa = linhaDePesquisaService.getAllFetch();
            } catch (Exception ex) {
```

```
                Utils.addExceptionMessage("Erro ao carregar Linhas de Pesquisa.");
                return new ArrayList<LinhaDePesquisa>(0);
            }
        }
        return linhasDePesquisa;
    }

    public Professor getSelectedProfessor() {
        return selectedProfessor;
    }

    public void setSelectedProfessor(Professor selectedProfessor) {
        this.selectedProfessor = selectedProfessor;
    }

    public void onTabChange(TabChangeEvent event) {
        Integer id = Integer.parseInt(event.getTab().getId().replace("linha", ""));
        LinhaDePesquisa linha = new LinhaDePesquisa();
        linha.setId(id);
        projetos = projetoService.getByLinhaDePesquisa(linha);
    }

    public List<Projeto> getProjetos() {
        return projetos;
    }

    public Projeto getSelectedProjeto() {
        return selectedProjeto;
    }

    public void setSelectedProjeto(Projeto selectedProjeto) {
        this.selectedProjeto = selectedProjeto;
    }
}
```

LoginBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Usuario;
import br.ufsc.inf.ppgcc.security.Acesso;
import br.ufsc.inf.ppgcc.service.UsuarioService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.io.IOException;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
```

```
@Controller
@Scope("session")
@Qualifier("loginBean")
public class LoginBean implements java.io.Serializable {

    @Autowired
    private UsuarioService usuarioService;
    private Usuario usuario;
    private String username;
    private String password;

    public void doLogin() throws ServletException, IOException {

        ExternalContext context = FacesContext.getCurrentInstance().getExternalContext();

        final ServletRequest request = (ServletRequest) context.getRequest();

        RequestDispatcher dispatcher =
request.getRequestDispatcher("//j_spring_security_check?j_username=" + username +
"&j_password=" + password);
        dispatcher.forward(request, (ServletResponse) context.getResponse());
        FacesContext.getCurrentInstance().responseComplete();

        if (this.isAutenticado()) {
            try {
                usuario = usuarioService.getByLogin(this.getAuthentication().getName());
            } catch (Exception ex) {
                Utils.addExceptionMessage("Erro ao carregar usuário.");
            }
        }

        username = null;
        password = null;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }
```

```
    public boolean isAutenticado() {
        Authentication a = this.getAuthentication();
        return !(a == null || a instanceof AnonymousAuthenticationToken);
    }

    public boolean isAdministrador() {
        return this.verificaAcesso(Acesso.ADMINISTRADOR);
    }

    public boolean isLeitura() {
        return this.verificaAcesso(Acesso.LEITURA);
    }

    public boolean isCoordenador() {
        return this.verificaAcesso(Acesso.COORDENADOR);
    }

    public boolean isProfessor() {
        return this.verificaAcesso(Acesso.PROFESSOR);
    }

    public boolean isAluno() {
        return this.verificaAcesso(Acesso.ALUNO);
    }

    private Authentication getAuthentication() {
        return SecurityContextHolder.getContext().getAuthentication();
    }

    private boolean verificaAcesso(GrantedAuthority acesso) {
        Authentication authentication = this.getAuthentication();
        return authentication != null && authentication.getAuthorities().contains(acesso) ? true :
false;
    }

    public boolean isUsuarioProfessor() {
        return usuario != null && usuario.isAssociadoAProfessor();
    }
}
```

OrientacoesBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.mail.MailUtil;
import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.Banca;
import br.ufsc.inf.ppgcc.pojo.BancaId;
import br.ufsc.inf.ppgcc.pojo.Defesa;
import br.ufsc.inf.ppgcc.pojo.Orientacao;
import br.ufsc.inf.ppgcc.pojo.OrientacaoId;
import br.ufsc.inf.ppgcc.pojo.PlanoDeTrabalho;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.recommendation.RecomendacaoPessoa;
import br.ufsc.inf.ppgcc.recommendation.SistemaDeRecomendacao;
import br.ufsc.inf.ppgcc.service.AlunoService;
import br.ufsc.inf.ppgcc.service.BancaService;
import br.ufsc.inf.ppgcc.service.OrientacaoService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
```

```
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("orientacoesBean")
public class OrientacoesBean implements java.io.Serializable {

    @Autowired
    private ProfessorService professorService;
    @Autowired
    private BancaService bancaService;
    @Autowired
    private AlunoService alunoService;
    @Autowired
    private OrientacaoService orientacaoService;
    private Professor professor;
    private Orientacao orientacaoTemp;
    private List<Professor> professoresSelect;
    private Set<Banca> bancas;
    private Banca banca;
    private Aluno alunoBanca;
    private Aluno alunoOrientar;
    private Professor professorBanca;
    private PlanoDeTrabalho selectedPlanoDeTrabalho;
    List<RecomendacaoPessoa<Aluno>> recomendacoes = new
ArrayList<RecomendacaoPessoa<Aluno>>(0);

    @Autowired
    public OrientacoesBean(LoginBean loginBean) {
        if (loginBean.getUsuario() != null) {
            professor = loginBean.getUsuario().getProfessor();
        }
    }

    public void recomendarAlunos() {
        try {
            List<Aluno> alunos = alunoService.getComProfessor(professor);

alunos.addAll(alunoService.getComLinhaDePesquisa(professor.getLinhasDePesquisa()));

            recomendacoes = SistemaDeRecomendacao.recomendarAlunos(professor, alunos);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao recomendar alunos.");
        }
    }

    public List<Orientacao> getOrientacoesByProfesor() {
        try {
            return orientacaoService.getByProfessor(professor);
        } catch (Exception e) {
```

```
            Utils.addExceptionMessage("Erro ao buscar orientações." + e.toString());
            return new ArrayList<Orientacao>(0);
        }
    }

    public void orientarAluno() {
        try {
            if (professor.getGrupo().getNumeroDeOrientacoes() >
orientacaoService.getNumOrientacoesEmAndamentoByProfessor(professor)) {
                Orientacao orientacao = new Orientacao();
                orientacao.setId(new OrientacaoId(alunoOrientar.getId(), professor.getId()));
                orientacao.setAluno(alunoOrientar);
                orientacao.setProfessor(professor);
                orientacao.setAceito(Boolean.TRUE);
                orientacao.setDataInicio(new Date());

                Defesa defesa = new Defesa();
                defesa.setIdAluno(alunoOrientar.getId());
                orientacao.getAluno().setDefesa(defesa);
                orientacao.setAceito(Boolean.TRUE);
                orientacao.setConvite(null);

                orientacaoService.save(orientacao);
                Utils.addMessage("Aluno pego para orientação com sucesso.");
            } else {
                Utils.addWarningMessage("O limite de orientações para o seu grupo foi atingindo.");
            }
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao orientar aluno." + e.toString());
        }
    }

    public void aceitarConvite() {
        try {
            Defesa defesa = new Defesa();
            defesa.setIdAluno(orientacaoTemp.getAluno().getId());
            orientacaoTemp.getAluno().setDefesa(defesa);

            orientacaoTemp.setAceito(Boolean.TRUE);
            orientacaoTemp.setConvite(null);
            orientacaoTemp.setDataInicio(Calendar.getInstance().getTime());
            professorService.save(professor);
            MailUtil.getInstance().enviarEmailConviteAceito(professor, orientacaoTemp.getAluno());
            Utils.addMessage("Convite de orientação aceito.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao aceitar convite.");
            orientacaoTemp.setAceito(null);
            orientacaoTemp.setConvite(Boolean.TRUE);
            orientacaoTemp.setDataInicio(null);
        }
    }

    public void liberarAluno() {
        alunoOrientar.getDadosInscricao().setProfessor(null);
        try {
            alunoService.save(alunoOrientar);
            Utils.addMessage("Aluno liberado com sucesso.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao liberar aluno." + e.toString());
```

```
    }
}

public List<Aluno> getAlunosComProfessor() {
    try {
        return alunoService.getComProfessor(professor);
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro ao buscar alunos." + e.toString());
        return new ArrayList<Aluno>(0);
    }
}

public List<Aluno> getAlunosComLinhaDePesquisa() {
    try {
        return alunoService.getComLinhaDePesquisa(professor.getLinhasDePesquisa());
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro ao buscar alunos." + e.toString());
        return new ArrayList<Aluno>(0);
    }
}

public void addProfessorBanca() {
    Banca b = new Banca();
    try {
        BancaId bancaId = new BancaId();
        bancaId.setDefesa(alunoBanca.getDefesa().getIdAluno());
        bancaId.setProfessor(professorBanca.getId());

        b.setDefesa(alunoBanca.getDefesa());
        b.setProfessor(professorBanca);
        b.setId(bancaId);
        bancaService.save(b);
        bancas.add(b);

        Utils.addMessage("Professor adicionado na banca com sucesso.");
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro ao adicionar professor na banca." + e);
    }
}

public void removeProfessorBanca() {
    try {
        bancaService.delete(banca);
        bancas.remove(banca);
        Utils.addMessage("Professor removido da banca com sucesso.");
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro ao remover professor da banca.");
    }
}

public void aceitarConviteBanca() {
    try {
        banca.setStatus(Banca.ACEITO_PROFESSOR);
        bancaService.update(banca);
        Utils.addMessage("Convite aceito com sucesso.");
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro aceitar convite.");
    }
}
```

```
    public void recusarConviteBanca() {
        try {
            banca.setStatus(Banca.RECUSADO_PROFESSOR);
            bancaService.update(banca);
            Utils.addMessage("Convite recusado com sucesso.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao recusar convite.");
        }
    }

    public List<Banca> getBancasComPendencia() {
        try {
            return bancaService.getBancasByStatus(Banca.ACEITO_COMITE, professor);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores com banca pendente." +
e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public List<Banca> getBancasAceitas() {
        try {
            return bancaService.getBancasByStatus(Banca.ACEITO_PROFESSOR, professor);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores com banca aceita." +
e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public List<Banca> getBancasRecusadas() {
        try {
            return bancaService.getBancasByStatus(Banca.RECUSADO_PROFESSOR,
professor);
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao buscar professores com banca recusada." +
e.toString());
            return new ArrayList<Banca>(0);
        }
    }

    public void recusarConvite() {
        try {
            orientacaoTemp.setAceito(Boolean.FALSE);
            orientacaoTemp.setConvite(null);
            professorService.save(professor);
            MailUtil.getInstance().enviarEmailConviteRejeitado(professor,
orientacaoTemp.getAluno());
            Utils.addMessage("Convite de orientação recusado.");
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao recusar convite.");
            orientacaoTemp.setAceito(null);
            orientacaoTemp.setConvite(Boolean.TRUE);
        }
    }

    public List<RecomendacaoPessoa<Aluno>> getRecomendacoes() {
        return recomendacoes;
```

```
}

public Professor getProfessor() {
    return professor;
}

public Professor getProfessorBanca() {
    return professorBanca;
}

public void setProfessorBanca(Professor professorBanca) {
    this.professorBanca = professorBanca;
}

public Orientacao getOrientacaoTemp() {
    return orientacaoTemp;
}

public void setOrientacaoTemp(Orientacao orientacaoTemp) {
    this.orientacaoTemp = orientacaoTemp;
}

public List<Professor> getProfessoresSelect() {
    try {
        professoresSelect = professorService.getAll();
    } catch (Exception e) {
        Utils.addExceptionMessage("Erro ao carregar professores.");
    }

    return professoresSelect;
}

public Set<Banca> getBancas() {
    return bancas;
}

public void setBancas(Set<Banca> bancas) {
    this.bancas = bancas;
}

public Banca getBanca() {
    return banca;
}

public void setBanca(Banca banca) {
    this.banca = banca;
}

public Aluno getAlunoBanca() {
    return alunoBanca;
}

public void setAlunoBanca(Aluno alunoBanca) {
    this.alunoBanca = alunoBanca;
}

public Aluno getAlunoOrientar() {
    return alunoOrientar;
}
```

```
    public void setAlunoOrientar(Aluno alunoOrientar) {
        this.alunoOrientar = alunoOrientar;
    }

    public PlanoDeTrabalho getSelectedPlanoDeTrabalho() {
        return selectedPlanoDeTrabalho;
    }

    public void setSelectedPlanoDeTrabalho(PlanoDeTrabalho selectedPlanoDeTrabalho) {
        this.selectedPlanoDeTrabalho = selectedPlanoDeTrabalho;
    }

    public void setProfessoresSelect() {
        try {
            professoresSelect = professorService.getAll();
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao carregar professores.");
        }
    }
}
```

PerfilBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Projeto;
import br.ufsc.inf.ppgcc.pojo.Usuario;
import br.ufsc.inf.ppgcc.pojo.datamodel.ProjetoDataModel;
import br.ufsc.inf.ppgcc.service.AlunoService;
import br.ufsc.inf.ppgcc.service.LinhaDePesquisaService;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.service.ProjetoService;
import br.ufsc.inf.ppgcc.service.UsuarioService;
import br.ufsc.inf.ppgcc.util.EditSaveCancel;
import br.ufsc.inf.ppgcc.util.NewEditDeleteSaveCancel;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import org.primefaces.event.SelectEvent;
import org.primefaces.event.UnselectEvent;
import org.primefaces.model.DualListModel;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("perfilBean")
public class PerfilBean implements java.io.Serializable {

    @Autowired
    private UsuarioService usuarioService;
    @Autowired
```

```
private ProfessorService professorService;
@Autowired
private LinhaDePesquisaService linhaDePesquisaService;
@Autowired
private AlunoService alunoService;
private ProjetoService projetoService;
private Usuario usuario;
private DualListModel<Professor> professores;
private LinhaDePesquisa linhaTemp;
private Professor professorTemp;
private Aluno alunoTemp;
private Projeto selectedProjeto;
private Projeto projetoTemp;
private List<Projeto> projetos;
private String senha1;
private String senha2;
/**
 * Controladores dos Botoes
 */
@Autowired
private EditSaveCancel botoesLinha;
@Autowired
private EditSaveCancel botoesProfessor;
@Autowired
private EditSaveCancel botoesAluno;
@Autowired
private EditSaveCancel botoesLogin;
@Autowired
private NewEditDeleteSaveCancel botoesProjeto;

@Autowired
public PerfilBean(LoginBean loginBean, ProjetoService projetoService) {
    usuario = loginBean.getUsuario();

    if (isAssociadoProfessor()) {
        try {
            projetos = projetoService.getByProfessor(usuario.getProfessor());
            if (projetos == null) {
                projetos = new ArrayList<Projeto>();
            }
        } catch (Exception e) {
            Utils.addExceptionMessage("Erro ao carregar projetos.");
        }
    }

    this.projetoService = projetoService;
}

public void botaoEditarLinha() {
    linhaTemp = usuario.getProfessor().getCoordena().clone();
    botoesLinha.controlarBotaoEditar();
}

public void botaoSalvarLinha() {
    try {
        linhaTemp.setProfessores(new HashSet<Professor>(professores.getTarget()));
        usuario.getProfessor().setCoordena(linhaTemp);
        linhaDePesquisaService.save(linhaTemp);
        Utils.addMessage("Linha de pesquisa salva com sucesso.");
```

```
            botoesLinha.controlarBotaoSalvar();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar linha de pesquisa." + ex.toString());
        }
    }

    public void botaoCancelarLinha() {
        botoesLinha.controlarBotaoCancelar();
    }

    public void botaoEditarProfessor() {
        professorTemp = usuario.getProfessor().clone();
        botoesProfessor.controlarBotaoEditar();
    }

    public void botaoSalvarProfessor() {
        try {
            usuario.setProfessor(professorTemp);
            professorService.save(professorTemp);
            Utils.addMessage("Professor salvo com sucesso.");
            botoesProfessor.controlarBotaoSalvar();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar professor." + ex.toString());
        }
    }

    public void botaoCancelarProfessor() {
        botoesProfessor.controlarBotaoCancelar();
    }

    public void botaoEditarAluno() {
        alunoTemp = usuario.getAluno().clone();
        botoesAluno.controlarBotaoEditar();
    }

    public void botaoSalvarAluno() {
        try {
            usuario.setAluno(alunoTemp);
            alunoService.save(alunoTemp);
            Utils.addMessage("Aluno salvo com sucesso.");
            botoesAluno.controlarBotaoSalvar();
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar aluno." + ex.toString());
        }
    }

    public void botaoCancelarAluno() {
        botoesAluno.controlarBotaoCancelar();
    }

    public void botaoEditarLogin() {
        botoesLogin.controlarBotaoEditar();
    }

    public void botaoSalvarLogin() {
        if (senha1.length() < 6) {
            Utils.addSpecificExceptionMessage("pitPerfilUsuarioSenha1", "A senha deve possuir
pelo menos 6 caracteres.");
        } else if (!senha1.equals(senha2)) {
```

```
            Utils.addSpecificExceptionMessage("pitPerfilUsuarioSenha1", "As senhas nÃ£o
conferem.");
        } else {
            try {
                usuario.setSenha(Utils.criptografarSenha(senha1));
                usuarioService.save(usuario);
                Utils.addMessage("Usuario salvo com sucesso.");
                botoesLogin.controlarBotaoSalvar();
                senha1 = null;
                senha2 = null;
            } catch (Exception ex) {
                Utils.addExceptionMessage("Erro ao salvar usuÃ¡rio." + ex.toString());
            }
        }
    }

    public void botaoCancelarLogin() {
        senha1 = null;
        senha2 = null;
        botoesLogin.controlarBotaoCancelar();
    }

    public void onSelectProjeto(SelectEvent event) {
        projetoTemp = this.getSelectedProjeto().clone();
        botoesProjeto.controlarOnSelect();
    }

    public void onUnselectProjeto(UnselectEvent event) {
        projetoTemp = new Projeto();
        botoesProjeto.controlarOnUnselect();
    }

    public void botaoNovoProjeto() {
        projetoTemp = new Projeto();
        botoesProjeto.controlarBotaoNovo();
    }

    public void botaoEditarProjeto() {
        botoesProjeto.controlarBotaoEditar();
    }

    public void botaoExcluirProjeto() {
        try {
            projetoService.delete(selectedProjeto);
            projetos.remove(this.getSelectedProjeto());
            selectedProjeto = null;
            projetoTemp = new Projeto();
            botoesProjeto.controlarBotaoExcluir();
            Utils.addMessage("Projeto excluÃ-do com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao excluir projeto." + ex.toString());
        }
    }

    public void botaoSalvarProjeto() {
        try {
            projetoTemp.setProfessor(usuario.getProfessor());
            projetoService.save(projetoTemp);
```

```
            if (botoesProjeto.isModoEditar()) {
                this.getSelectedProjeto().copy(projetoTemp);
            } else if (botoesProjeto.isModoNovo()) {
                projetos.add(projetoTemp);
                usuario.getProfessor().getProjetos().add(projetoTemp);
            }

            botoesProjeto.controlarBotaoSalvar();
            Utils.addMessage("Projeto salvo com sucesso.");
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao salvar projeto." + ex.toString());
        }
    }

    public void botaoCancelarProjeto() {
        projetoTemp = this.getSelectedProjeto().clone();
        if (selectedProjeto == null) {
            botoesProjeto.controlarOnUnselect();
        } else {
            botoesProjeto.controlarBotaoCancelar();
        }
    }

    public boolean isAssociadoAluno() {
        return usuario.isAssociadoAAluno();
    }

    public final boolean isAssociadoProfessor() {
        return usuario.isAssociadoAProfessor();
    }

    public boolean isUsuarioCoordenador() {
        return usuario.isAssociadoACoordenador();
    }

    /*
     * Getters and Setters
     *
     */
    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public DualListModel<Professor> getProfessores() {
        try {
            List<Professor> professoresDisponiveis = professorService.getAll();
            List<Professor> professoresLinha =
usuario.getProfessor().getCoordena().getProfessoresList();
            professoresDisponiveis.removeAll(professoresLinha);
            professores = new DualListModel<Professor>(professoresDisponiveis,
professoresLinha);
        } catch (Exception ex) {
            Utils.addExceptionMessage("Erro ao carregar Professores." + ex.toString());
        }
```

```
    return professores;
}

public void setProfessores(DualListModel<Professor> professores) {
    this.professores = professores;
}

public Aluno getAlunoTemp() {
    return alunoTemp;
}

public void setAlunoTemp(Aluno alunoTemp) {
    this.alunoTemp = alunoTemp;
}

public LinhaDePesquisa getLinhaTemp() {
    return linhaTemp;
}

public void setLinhaTemp(LinhaDePesquisa linhaTemp) {
    this.linhaTemp = linhaTemp;
}

public Professor getProfessorTemp() {
    return professorTemp;
}

public void setProfessorTemp(Professor professorTemp) {
    this.professorTemp = professorTemp;
}

public String getSenha1() {
    return senha1;
}

public void setSenha1(String senha1) {
    this.senha1 = senha1;
}

public String getSenha2() {
    return senha2;
}

public void setSenha2(String senha2) {
    this.senha2 = senha2;
}

public EditSaveCancel getBotoesAluno() {
    return botoesAluno;
}

public EditSaveCancel getBotoesLinha() {
    return botoesLinha;
}

public EditSaveCancel getBotoesLogin() {
    return botoesLogin;
}
```

```
    public EditSaveCancel getBotoesProfessor() {
        return botoesProfessor;
    }

    public NewEditDeleteSaveCancel getBotoesProjeto() {
        return botoesProjeto;
    }

    public Projeto getSelectedProjeto() {
        return selectedProjeto != null ? selectedProjeto : new Projeto();
    }

    public void setSelectedProjeto(Projeto selectedProjeto) {
        this.selectedProjeto = selectedProjeto;
    }

    public Projeto getProjetoTemp() {
        return projetoTemp;
    }

    public void setProjetoTemp(Projeto projetoTemp) {
        this.projetoTemp = projetoTemp;
    }

    public ProjetoDataModel getProjetos() {
        return new ProjetoDataModel(projetos);
    }
}
```

ProfessoresBean

```
package br.ufsc.inf.ppgcc.controller;

import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.service.ProfessorService;
import br.ufsc.inf.ppgcc.util.Utils;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("view")
@Qualifier("professoresBean")
public class ProfessoresBean {

    @Autowired
    private ProfessorService professorService;
    private List<Professor> professores;
    private LinhaDePesquisa selectedLinhaDePesquisa;

    public List<Professor> getProfessores() {
        if (professores == null) {
            try {
                professores = professorService.getAllFetch();
            } catch (Exception ex) {
```

```
            Utils.addExceptionMessage("Erro ao carregar Professores.");
            return new ArrayList<Professor>(0);
          }
        }
        return professores;
    }

    public LinhaDePesquisa getSelectedLinhaDePesquisa() {
        return selectedLinhaDePesquisa;
    }

    public void setSelectedLinhaDePesquisa(LinhaDePesquisa selectedLinhaDePesquisa) {
        this.selectedLinhaDePesquisa = selectedLinhaDePesquisa;
    }
}
```

Recomendacao

```
package br.ufsc.inf.ppgcc.recommendation;

public class Recomendacao<T> implements Comparable<Recomendacao> {

    private float score;
    private T recomendado;

    public T getRecomendado() {
        return recomendado;
    }

    public void setRecomendado(T recomendado) {
        this.recomendado = recomendado;
    }

    public float getScore() {
        return score;
    }

    public void setScore(float score) {
        this.score = score;
    }

    public int compareTo(Recomendacao o) {
        return score > o.score ? -1 : score < o.score ? 1 : 0;
    }
}
```

RecomendacaoPessoa

```
package br.ufsc.inf.ppgcc.recommendation;

import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Projeto;
import java.util.List;

public class RecomendacaoPessoa<T> implements Comparable<RecomendacaoPessoa> {

    private T recomendado;
    private float mediaScore;
    private List<Recomendacao<LinhaDePesquisa>> recomendacaoLinhas;
```

```
    private List<Recomendacao<Projeto>> recomendacaoProjetos;

    public T getRecomendado() {
        return recomendado;
    }

    public void setRecomendado(T recomendado) {
        this.recomendado = recomendado;
    }

    public float getMediaScore() {
        return mediaScore;
    }

    public void setMediaScore(float mediaScore) {
        this.mediaScore = mediaScore;
    }

    public List<Recomendacao<LinhaDePesquisa>> getRecomendacaoLinhas() {
        return recomendacaoLinhas;
    }

    public void setRecomendacaoLinhas(List<Recomendacao<LinhaDePesquisa>>
recomendacaoLinhas) {
        this.recomendacaoLinhas = recomendacaoLinhas;
    }

    public List<Recomendacao<Projeto>> getRecomendacaoProjetos() {
        return recomendacaoProjetos;
    }

    public void setRecomendacaoProjetos(List<Recomendacao<Projeto>>
recomendacaoProjetos) {
        this.recomendacaoProjetos = recomendacaoProjetos;
    }

    public int compareTo(RecomendacaoPessoa o) {
        return mediaScore > o.mediaScore ? -1 : mediaScore < o.mediaScore ? 1 : 0;
    }
}
```

SistemaDeRecomendacao

```
package br.ufsc.inf.ppgcc.recommendation;

import br.ufsc.inf.ppgcc.pojo.Aluno;
import br.ufsc.inf.ppgcc.pojo.LinhaDePesquisa;
import br.ufsc.inf.ppgcc.pojo.Professor;
import br.ufsc.inf.ppgcc.pojo.Projeto;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import uk.ac.shef.wit.simmetrics.similaritymetrics.JaccardSimilarity;
import uk.ac.shef.wit.simmetrics.similaritymetrics.QGramsDistance;
import uk.ac.shef.wit.simmetrics.tokenisers.InterfaceTokeniser;
import uk.ac.shef.wit.simmetrics.tokenisers.TokeniserWhitespace;
import uk.ac.shef.wit.simmetrics.wordhandlers.GenericStopTermHandler;
import uk.ac.shef.wit.simmetrics.wordhandlers.InterfaceTermHandler;
```

```
public class SistemaDeRecomendacao {

    private static final QGramsDistance Q_GRAMS_DISTANCE;
    private static final InterfaceTokeniser tokeniser;
    private static final InterfaceTermHandler termHandler;
    private static final String[] stopWords;
    private static final String[] characters;

    static {
        stopWords = new String[]{
            "a", "à", "agora", "ainda", "alguém", "algum", "alguma", "algumas", "alguns", "ampla",
            "amplas", "amplo", "amplos", "ante", "antes", "ao", "aos", "após", "aquela", "aquelas",
            "aquele", "aqueles", "aquilo", "as", "até", "através", "cada", "coisa", "coisas", "com",
            "como", "contra", "contudo", "da", "daquele", "daqueles", "das", "de", "dela", "delas",
            "dele", "deles", "depois", "dessa", "dessas", "desse", "desses", "desta", "destas",
            "deste", "deste", "destes", "deve", "devem", "devendo", "dever", "deverá", "deverão",
            "deveria", "deveriam", "devia", "deviam", "disse", "disso", "disto", "dito", "diz",
            "dizem", "do", "dos", "e", "é", "e'", "ela", "elas", "ele", "eles", "em", "enquanto",
            "entre", "era", "essa", "essas", "esse", "esses", "esta", "está", "estamos", "estão",
            "estas", "estava", "estavam", "estávamos", "este", "estes", "estou", "eu", "fazendo",
            "fazer", "feita", "feitas", "feito", "feitos", "foi", "for", "foram", "fosse", "fossem",
            "grande", "grandes", "há", "isso", "isto", "já", "la", "la", "lá", "lhe", "lhes",
            "lo", "mas", "me", "mesma", "mesmas", "mesmo", "mesmos", "meu", "meus", "minha",
            "minhas", "muita", "muitas", "muito", "muitos", "na", "não", "nas", "nem", "nenhum",
            "nessa", "nessas", "nesta", "nestas", "ninguém", "no", "nos", "nós", "nossa", "nossas",
            "nosso", "nossos", "num", "numa", "nunca", "o", "os", "ou", "outra", "outras", "outro",
            "outros", "para", "pela", "pelas", "pelo", "pelos", "pequena", "pequenas", "pequeno",
            "pequenos", "per", "perante", "pode", "pôde", "podendo", "poder", "poderia", "poderiam",
            "podia", "podiam", "pois", "por", "porém", "porque", "posso", "pouca", "poucas", "pouco",
            "poucos", "primeiro", "primeiros", "própria", "próprias", "próprio", "próprios", "quais",
            "qual", "quando", "quanto", "quantos", "que", "quem", "são", "se", "seja", "sejam",
            "sem", "sempre", "sendo", "será", "serão", "seu", "seus", "si", "sido", "só", "sob",
            "sobre", "sua", "suas", "talvez", "também", "tampouco", "te", "tem", "tendo", "tenha",
            "ter", "teu", "teus", "ti", "tido", "tinha", "tinham", "toda", "todas", "todavia", "todo",
            "todos", "tu", "tua", "tuas", "tudo", "última", "últimas", "último", "últimos", "um",
            "uma", "umas", "uns", "vendo", "ver", "vez", "vindo", "vir", "vos", "vós"};

        characters = new String[]{
            "\\.", ",", "/", "\\", "'", "\"", ";", ":", "!", "\\?",
            "\\(", "\\)", "\\[", "]", "\\{", "}", "\\|", "\\*"};

        termHandler = new GenericStopTermHandler();
        for (String word : stopWords) {
            termHandler.addWord(word.toUpperCase());
        }

        tokeniser = new TokeniserWhitespace();
        tokeniser.setStopWordHandler(termHandler);

        Q_GRAMS_DISTANCE = new QGramsDistance(tokeniser);
    }

    public static List<RecomendacaoPessoa<Professor>> recomendarProfessores(Aluno aluno,
List<Professor> professores) {
        List<RecomendacaoPessoa<Professor>> recomendacoes = new
ArrayList<RecomendacaoPessoa<Professor>>();

        for (Professor professor : professores) {
```

```
            RecomendacaoPessoa recomendacaoDeProfessor =
getRecomendacaoDeProfessor(aluno, professor);

            if (recomendacaoDeProfessor.getMediaScore() > 0.0) {
                recomendacoes.add(recomendacaoDeProfessor);
            }
        }

        Collections.sort(recomendacoes);

        return recomendacoes.subList(0, recomendacoes.size() < 5 ? recomendacoes.size() : 5);
    }

    public static List<RecomendacaoPessoa<Aluno>> recomendarAlunos(Professor professor,
List<Aluno> alunos) {
        List<RecomendacaoPessoa<Aluno>> recomendacoes = new
ArrayList<RecomendacaoPessoa<Aluno>>();

        for (Aluno aluno : alunos) {
            RecomendacaoPessoa recomendacaoDeAluno =
getRecomendacaoDeAluno(professor, aluno);

            if (recomendacaoDeAluno.getMediaScore() > 0.0) {
                recomendacoes.add(recomendacaoDeAluno);
            }
        }

        Collections.sort(recomendacoes);

        return recomendacoes.subList(0, recomendacoes.size() < 5 ? recomendacoes.size() : 5);
    }

    private static RecomendacaoPessoa getRecomendacaoDeProfessor(Aluno aluno, Professor
professor) {
        RecomendacaoPessoa<Professor> recomendacao = new
RecomendacaoPessoa<Professor>();
        recomendacao.setRecomendado(professor);

        setRecomendacaoPessoa(recomendacao, aluno, professor);

        return recomendacao;
    }

    private static RecomendacaoPessoa getRecomendacaoDeAluno(Professor professor, Aluno
aluno) {
        RecomendacaoPessoa<Aluno> recomendacao = new RecomendacaoPessoa<Aluno>();
        recomendacao.setRecomendado(aluno);

        setRecomendacaoPessoa(recomendacao, aluno, professor);

        return recomendacao;
    }

    private static void setRecomendacaoPessoa(RecomendacaoPessoa<?> recomendacao,
Aluno aluno, Professor professor) {
        List<Recomendacao<LinhaDePesquisa>> recomendacaoLinhas = new
ArrayList<Recomendacao<LinhaDePesquisa>>();
        List<Recomendacao<Projeto>> recomendacaoProjetos = new
ArrayList<Recomendacao<Projeto>>();
```

```
        recomendacao.setRecomendacaoLinhas(recomendacaoLinhas);
        recomendacao.setRecomendacaoProjetos(recomendacaoProjetos);

        String textAluno = aluno.getPlanoDeTrabalho().getPlano();
        float score = 0.0f;

        Recomendacao<LinhaDePesquisa> recLinha;
        for (LinhaDePesquisa linha : professor.getLinhasDePesquisa()) {
            float similaridade = getSimilaridade(textAluno, linha.getObjetivos());
            score += similaridade;

            recLinha = new Recomendacao<LinhaDePesquisa>();
            recLinha.setRecomendado(linha);
            recLinha.setScore(similaridade);
            recomendacaoLinhas.add(recLinha);
        }

        Recomendacao<Projeto> recProjeto;
        for (Projeto projeto : professor.getProjetos()) {
            float similaridade = getSimilaridade(textAluno, projeto.getDescricao());
            score += similaridade;

            recProjeto = new Recomendacao<Projeto>();
            recProjeto.setRecomendado(projeto);
            recProjeto.setScore(similaridade);
            recomendacaoProjetos.add(recProjeto);
        }

        Collections.sort(recomendacaoLinhas);
        Collections.sort(recomendacaoProjetos);

        recomendacao.setMediaScore(score / (professor.getLinhasDePesquisa().size() +
professor.getProjetos().size()));
    }

    private static float getSimilaridade(String text1, String text2) {
        for (String ch : characters) {
            text1 = text1.replaceAll(ch, "");
            text2 = text2.replaceAll(ch, "");
        }

        return Q_GRAMS_DISTANCE.getSimilarity(text1.toUpperCase(), text2.toUpperCase());
    }
}
```