UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

JOÃO GUILHERME BRASIL PICHETTI

APLICATIVO WEB PARA BANCAS DE AVALIAÇÃO DE TRABALHOS ACADÊMICOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

JOÃO GUILHERME BRASIL PICHETTI

APLICATIVO WEB PARA AVALIAÇÃO DE TRABALHOS ACADÊMICOS

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2015

APLICATIVO WEB PARA BANCAS DE AVALIAÇÃO DE TRABALHOS ACADÊMICOS

Por

João Guilherme Brasil Pichetti

Esta monografia foi apresentada às 16h45 do dia 23 de setembro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profa. Dr. Beatriz Bisoi
Orientadora
UTFPR – Campus Pato Branco

Prof. Msc. Vinicius Pegorini
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

RESUMO

PICHETTI, João Guilherme Brasil. Aplicativo web para avaliação de trabalhos acadêmicos. 2015. 62 f. Monografia (Trabalho de Conclusão de Curso) - Curso de Especialização em Tecnologia Java, Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

Na Universidade Tecnológica Federal do Paraná (UTFPR), as atividades acadêmicas como apresentações de estágio curricular, propostas e projetos de trabalho de conclusão de curso e monografias, dissertações e teses ocorrem perante uma banca. A avaliação do trabalho é realizada por uma banca composta por professores da própria instituição, de outras instituições e mesmo de pessoas da comunidade. A avaliação é realizada por meio de itens para os quais são atribuídas notas e cada item pode possuir um peso associado. Esses itens podem ser agrupados, compondo áreas de avaliação. Dessa avaliação é obtida uma nota (média ponderada se pesos atribuídos às notas dos itens) ou a aprovação ou não no caso de proposta de trabalho de conclusão de curso. Em determinados tipos de apresentação, uma ata é gerada para registrar a ocorrência da apresentação. Atualmente, no Departamento Acadêmico de Informática da UTFPR, Câmpus Pato Branco, a composição das bancas, a disponibilização dos trabalhos para os avaliadores, a avaliação dos trabalhos e a geração da ata e do relatório de participantes, são atividades realizadas ainda sem o suporte de um sistema informatizado. Como resultado deste trabalho, um aplicativo computacional desenvolvido em Java para *web*, foi implementado visando automatizar várias tarefas envolvidas na realização das bancas e avaliação dos trabalhos.

Palavras-chave: Aplicativo web. Java para web. Aplicativo para avaliação de bancas de trabalhos acadêmicos.

ABSTRACT

PICHETTI, João Guilherme Brasil. Web system to evaluate of academic works. 2015. 62 f. Monografia (Trabalho de Conclusão de Curso) - Curso de Especialização em Tecnologia Java, Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

In the Federal Technological University of Paraná (UTFPR), academic activities such as traineeship reports, proposals of completion course, monographs, dissertations and theses are valued by presentation of a group of people. The evaluation is conducted by a panel composed of teachers and even people from the community (enterprises). The evaluation is carried out by items for which notes are assigned and each item may have an associated weight. These items can be grouped, making assessment areas. From this evaluation is obtained a note (weighted average are assigned to the notes of the items) or approval or not, in the case of proposal of completion course works. In some types of presentation, academic-figure is generated to record the occurrence of the presentation. Currently, in the Academic Department of Informatics of UTFPR, Campus Pato Branco, the activities related to these academic works are still carried out without the support of a system, such as software. As a result of this work, a computer application developed in Java for web has been implemented aiming to help the fulfillment of various tasks involved in these academics presentation.

Keywords: Web application. Web Java. Application for assessment of academic work.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

CSS	*Cascading Style Sheets*
CRUD	*Create-Read-Update-Delete*
DAO	*Data Access Object*
HTML	*HiperText Markup Language*
IDE	*Integrated Develpment Environmet*
JPA	*Java Persistence*
JPQL	*Java Persistence Query Language*
JSF	*Java Server Faces*
JSR	*Java Specification Requests*
ORM	*Object Relational Mapping*
POM	*Project Object Model*
RIA	*Rich Internet Application*
SQL	*Structured Query Language*
UTFPR	Universidade Tecnológica Federal do Paraná
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. No final do capítulo é apresentada a organização do texto por meio de uma breve descrição dos seus capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Atividades acadêmicas como, por exemplo, estágios curriculares, propostas e monografias de trabalhos de conclusão de curso são avaliadas por meio de apresentação realizada perante uma banca. As bancas são compostas por professores da própria Universidade, de outras Instituições de Ensino e mesmo de pessoas de fora do meio acadêmico. Um relatório de estágio, por exemplo, pode ter como membro de banca um profissional da empresa na qual o aluno realizou a atividade de estágio.

No Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Pato Branco, a avaliação desses trabalhos tem sido realizada por meio de fichas impressas ou planilhas de cálculo que são preenchidas pelos membros da banca. O professor responsável pela atividade de estágio ou trabalho de conclusão do curso, por exemplo, faz a composição das médias e emite a ata de defesa. A ata é composta por um texto padrão e por informações específicas de cada apresentação.

Diante desse contexto, percebeu-se que um sistema computacional que permita aos membros das bancas lançarem as notas das suas avaliações e que os itens dessas avaliações possam ser compostos, contribuirá para facilitar a realização dessa atividade. Um aplicativo *web* facilita o acesso pelos avaliadores e professor responsável pela atividade sendo avaliada.

Uma aplicação *web* apresenta recursos de interface que têm sido encontrados em aplicações *desktop* e que visa minimizar o tráfego de redes pela atualização de partes específicas da página e que provê, também, possibilidade de processamento e armazenamento de dados no cliente, é caracterizada como *Rich Internet Application* (RIA). O aplicativo desenvolvido como resultado da realização deste trabalho atende a essas características.

1.2 OBJETIVOS

O objetivo geral está relacionado ao resultado principal que é esperado da realização deste trabalho. E os objetivos específicos complementam o objetivo geral em termos de funcionalidades do sistema.

1.2.1 Objetivo Geral

Implementar um aplicativo para gerenciar avaliações de trabalhos acadêmicos que são apresentados perante bancas avaliadoras.

1.2.2 Objetivos Específicos

- Facilitar a composição dos itens para a avaliação pelos responsáveis pelas atividades acadêmicas de estágio e trabalho de conclusão de curso.
- Permitir ao aluno fazer *upload* do trabalho a ser avaliado e aos avaliadores de observações sobre o trabalho.
- Agilizar o processo de registro da avaliação de bancas de trabalhos acadêmicos por parte dos membros de bancas avaliadoras.

1.3 JUSTIFICATIVA

A falta de praticidade no uso de fichas impressas percebida por avaliadores e professores responsáveis por trabalhos como os de estágio, de conclusão de curso e de propostas de trabalhos de conclusão de curso é a justificativa principal para a realização deste trabalho. Para as pessoas que realizam a avaliação, a manipulação das fichas impressas nem sempre é eficiente. Para os professores responsáveis por essas atividades é necessário, a partir das fichas dos membros das bancas, calcular e lançar médias e elaborar as atas de defesa por meio de um aplicativo específico.

Para os professores participantes dessas bancas, um sistema *web* torna o processo de avaliação mais prático. Para os professores responsáveis pelas atividades, além de não ser necessário realizar cálculos e lançamentos, é muito mais prático compor as bancas, avisar os envolvidos e compor os itens para cada tipo de avaliação. Estando esses itens pré-cadastrados é possível vinculá-los em uma avaliação e atribuir pesos aos mesmos. Assim, para trabalhos distintos haverá avaliações compostas por seus itens específicos.

A justificativa de aplicabilidade do resultado deste trabalho se fundamenta na necessidade percebida de facilitar o processo de avaliação que é realizada pelos membros de bancas. Em termos de tecnologias, a escolha de implementação de um sistema para *web* decorre pela facilidade de acesso possibilitada por meio da Internet.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. Este é o primeiro e apresenta as considerações iniciais, o objetivo e a justificativa do trabalho. O Capítulo 2 apresenta o referencial teórico que se refere ao desenvolvimento de aplicações *web*. No Capítulo 3 estão os materiais e o método utilizados para a modelagem e a implementação do sistema. Os resultados da realização deste trabalho são apresentados no Capítulo 4. Por fim está a conclusão, seguida das referências bibliográficas.

2 DESENVOLVIMENTO DE APLICAÇÕES WEB

Este capítulo apresenta a fundamentação conceitual relacionada às tecnologias utilizadas para a implementação do sistema que é resultado deste trabalho. Essas tecnologias estão relacionadas ao conceito de aplicações Internet ricas, as RIAs.

2.1 CONTEXTO CONCEITUAL

A ampliação das exigências por parte dos usuários das aplicações Internet é representada por Dworak (2008) quando diz que já se foram os dias nos quais a *web* consistia basicamente de páginas (*home pages*) personalizadas caracterizando conteúdo estático. E com esse conteúdo organizado em conjuntos de quadros e contendo música de fundo, nem sempre de bom gosto.

A tendência atual é o desenvolvimento de aplicações de negócio para Internet com tecnologias relacionadas à *HiperText Markup Language* (HTML) (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). Porém, HTML foi originalmente criada como uma linguagem de marcação de hipertexto para Internet. HTML não permite recarregamento parcial da página. Contudo, para esses autores, com programação em JavaScript, por exemplo, elementos específicos da página podem ser manipulados.

Para Dworak (2008), aplicações *web* modernas têm impulsionado os navegadores *web* para além dos seus limites visando proporcionar uma boa experiência no uso de sistemas web para a ampla quantidade de usuários da Internet. Esses usuários possuem interesses e habilidades em lidar com recursos computacionais distintas. Dworak (2008) ressalta que essas aplicações são capazes de reduzir a latência, a largura de banda e os recursos utilizados do servidor. Isso é decorrente da requisição autônoma e da análise de dados armazenados em um servidor seguido pela atualização seletiva de regiões da página de uma forma semelhante às aplicações *desktop* tradicionais.

As *Rich Internet Applications* (RIA) são aplicações cliente/servidor que surgiram da intersecção do desenvolvimento das aplicações *web* e *desktop* (MELIA *et al*., 2010). As RIAs se beneficiam das melhores características dessas duas formas de aplicação distintas em termos da forma de acesso. Essas características são: distribuição do processamento entre cliente e servidor e manutenção da *web*, ao mesmo tempo em que suporta uma interface com o usuário mais rica, como as aplicações *desktop*.

A Figura 1 descreve a forma dde trabalho de uma RIA (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). A aplicação está localizada no servidor. A definição da interface gráfica com o usuário é transferida do servidor para o cliente. O cliente recebe a definição da interface e compõe a página para o usuário. Se o usuário realiza uma operação (como, clicar em um *link* existente na página), a informação sobre a ação do usuário é transferida por meio da rede de volta para a aplicação que está no servidor e então é executado o código vinculado à ação que deve ser realizada em resposta ao ocorrido na página. A definição da interface gráfica com o usuário, como abrir uma nova página, será transferida para o cliente e a ação será representada no cliente.

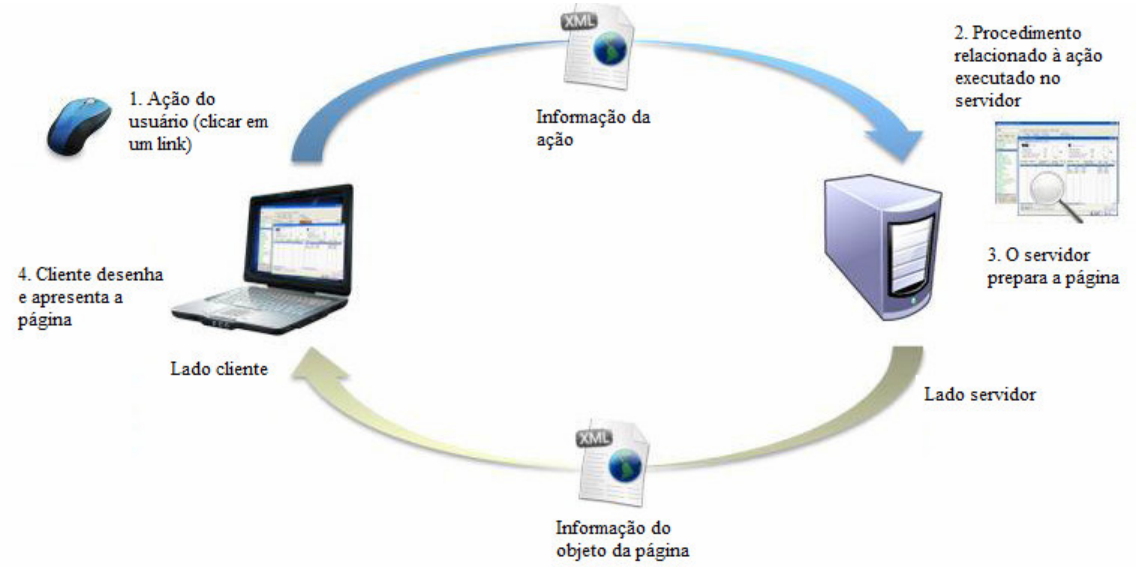

Figura 1 – Modelo de comunicação da RIA

Fonte: traduzido de Pavlić, Pavlić e Jovanović (2012, p. 1367).

As RIAs oferecem comportamento em tempo de execução mais flexível se comparado com aplicações *web* tradicionais. Para Comai e Carughi (2007), em relação à interação com o usuário, o cliente pode solicitar seletivamente requisições a partir do servidor somente de parte dos dados e manter inalterado o restante das informações que não são afetadas. Além disso, a interação com o usuário pode causar necessidade de alteração em apenas parte do conteúdo, que foi anteriormente apresentado pela aplicação, mas tornou-se inconsistente com o restante da página.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para a realização deste trabalho. Os materiais estão relacionados às tecnologias e às ferramentas utilizadas para modelar e implementar o aplicativo e o método apresenta a sequência das principais atividades realizadas para o levantamento dos requisitos, modelagem e implementação dos requisitos.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas para modelar e implementar o sistema desenvolvido.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Astah* Community	6.2.1 (model version: 33)	http://astah.net/editions/community	Documentação da modelagem baseada na UML.
Linguagem Java	JDK 1.8	http://www.oracle.com	Linguagem de programação.
Eclipse Mars	4.5	https://eclipse.org/mars/	Ambiente de desenvolvimento.
Apache Maven	3.3.3	https://maven.apache.org/	Ferramenta para gerenciamento do projeto.
MySQL	5.6	http://www.mysql.com/	Banco de dados.
MySQL WorkBench	6.1 CE	http://www.mysql.com/products/workbench/	Administrador do banco de dados.
PrimeFaces	5.2	http://primefaces.org/	Biblioteca de componentes para *web*.
JSF	2.2	https://javaserverfaces-spec-public.java.net/	*Framework* para *web.*
Spring Boot	1.2.4	http://projects.spring.io/spring-boot/	Framework para aumento de produtividade.

Quadro 1 – Ferramentas e tecnologias utilizadas

a) JavaServer Faces

O JSF é um *framework* Java para desenvolvimento de aplicações *web*, baseado em componentes, o que torna quase desnecessário o conhecimento em HTML, CSS e JavaScript no desenvolvimento de aplicações. Além disso, ele é formalizado como uma especificação e, assim, todo servidor de aplicações Java deve possuir uma implementação sua. Duas implementações conhecidas são: Oracle Mojarra (MOJARRA, 2015) e Apache MyFaces (MYFACES, 2015). Por ser uma especificação, ele contém apenas os componentes

fundamentais para atender a demanda por componentes sofisticados. Há várias extensões do JSF, três das mais conhecidas são: PrimeFaces (PRIMEFACES, 2015), RichFaces (RICHFACES, 2015) e IceFaces (ICEFACES, 2015).

b) Spring

O Spring é um *framework* criado com o intuito de simplificar a programação de aplicações Java, baseado no padrão de inversão de controle e injeção de dependências, ou seja, o desenvolvedor não precisa se preocupar com a instanciação de novos objetos, delegando ao *framework* que instancie o objeto apenas quando for necessário. Além disso, o Spring possui diversos módulos que podem ser utilizados conforme a necessidade do projeto, como módulo para persistência de dados, desenvolvimento *web* e segurança. Como ele é dividido em módulos, utiliza-se apenas o necessário, tornando a aplicação menor. Toda a configuração necessária pelo Spring é feita por meio de arquivos *Extensible Markup Language* (XML), tornando o entendimento do *framework* um pouco complexo para desenvolvedores iniciantes.

Já o Spring Boot é um projeto que engloba todos os módulos já existentes no Spring Framework, mas sua configuração é feita através de classes Java e anotações. Seu objetivo não é trazer novos módulos, mas sim reaproveitar os módulos já existentes aumentando a produtividade e melhorando o entendimento do *framework*.

3.2 MÉTODO

A seguir estão descritas as etapas definidas para o desenvolvimento do aplicativo e as principais atividades de cada uma dessas etapas. Uma versão inicial do sistema foi desenvolvida pelo autor deste trabalho. Assim, neste trabalho os requisitos foram revisados e complementados.

a) Requisitos

O levantamento dos requisitos foi realizado tendo como base as necessidades dos cursos do Departamento Acadêmico de Informática da UTFPR, Câmpus Pato Branco. Os docentes responsáveis pelas atividades de estágio e conclusão de curso desse Departamento forneceram os requisitos para o sistema. Assim, uma visão geral do sistema foi obtida, mas à medida que a modelagem e mesmo a implementação foram realizadas muitos complementos foram necessários. Algumas regras de negócio relacionadas à disponibilização do trabalho

para correção, por exemplo, se apresentaram bem mais complexas do que o inicialmente identificado.

b) Análise e projeto do sistema

A modelagem dos requisitos foi realizada visando complementar uma versão prévia do sistema que permitia a composição da banca. Os requisitos modelados e implementados neste trabalho permitem, além da avaliação do trabalho, a postagem do texto para ser disponibilizado para a banca e o acompanhamento da avaliação realizada pelos membros da banca.

c) Implementação

A implementação foi realizada utilizando a ferramenta Eclipse Mars e as tecnologias indicadas no Quadro 1.

d) Testes

Os testes foram informais e realizados à medida que as funcionalidades do sistema eram implementadas.

4 RESULTADO

Este capítulo apresenta o resultado deste trabalho que é a implementação de um aplicativo para que alunos postem trabalhos para avaliação e membros de banca realizar a avaliação dos trabalhos apresentados perante banca.

4.1 ESCOPO DO SISTEMA

O aplicativo computacional implementado como resultado deste trabalho visa auxiliar nas atividades de avaliação de trabalhos acadêmicos que são apresentados perante banca. Essas atividades envolvem, entre outras, a disponibilização do trabalho para a banca, a postagem de considerações de ajustes feitas pelos membros da banca (orientadores e avaliadores), a postagem da versão final e a postagem da ata de defesa, além do acesso ao formulário de avaliação que é realizada por cada um dos membros da banca. Como exemplos desses trabalhos estão os relatórios de estágio curricular, os trabalhos de conclusão de curso e as propostas e projetos de trabalhos de conclusão de curso. A solução proposta considera o contexto apresentado a seguir.

Um trabalho apresentado em banca pode ser realizado por mais de um aluno, embora o padrão seja um aluno. Se houver mais de um aluno vinculado a um mesmo trabalho, cada aluno é avaliado individualmente por cada um dos membros que compõem a banca. A avaliação é realizada por meio de itens que podem ter pesos associados. Neste caso, a nota do aluno é obtida pela média aritmética a partir da média ponderada de cada componente da banca. Cada tipo de trabalho terá banca composta por um número distinto de componentes, geralmente professores, mas podem ser profissionais, como no caso de avaliação de estágio curricular obrigatório, por exemplo. É comum a quantidade de componentes de uma banca ser definida de acordo com o tipo de trabalho.

O professor que é responsável pela atividade de estágio, de trabalho de conclusão de curso ou outra atividade que é avaliada por banca de avaliadores, faz a composição da banca para o respectivo trabalho. Esse responsável tem acesso somente aos trabalhos para os quais ele é o responsável. Além do tipo de atividade, o professor responsável está associado a um curso. Assim, por exemplo, o professor responsável pelo trabalho de conclusão de curso do curso de Tecnologia em Análise e Desenvolvimento de Sistemas terá acesso somente às bancas de trabalhos e suas respectivas propostas desse curso; o professor responsável pela

atividade de estágio do curso de Engenharia de Computação somente terá acesso aos trabalhos de estágio desse curso. Ressalta-se que o tipo de trabalho ou projeto (estágio, trabalho de conclusão de curso, proposta) é cadastrado, ou seja, não está predefinido no aplicativo.

O avaliador terá acesso somente aos trabalhos dos quais ele é membro de banca. Os trabalhos e respectivo formulário de avaliação são disponibilizados somente a partir da composição da banca. O arquivo para *download* depende, também, de ter sido postado no sistema pelo aluno ou respectivo orientador.

4.2 MODELAGEM DO SISTEMA

O diagrama conceitual de domínio é apresentado na Figura 2. Essa figura tem o objetivo de apresentar a visão geral do sistema representada como um conjunto de conceitos relacionados. Esses conceitos e os seus relacionamentos não necessariamente representam classes e/ou entidades de armazenamento de dados, mas fornecem uma ideia das principais funcionalidades do sistema.

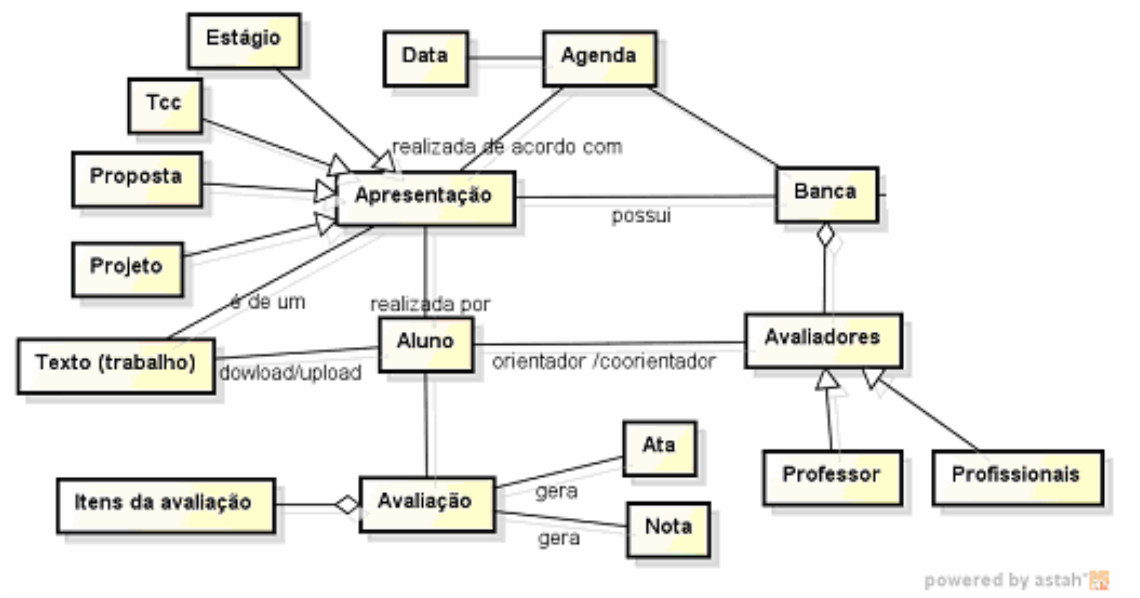

Figura 2 – Modelo conceitual de domínio do sistema

De acordo com a representação da Figura 2, uma apresentação pode ser de vários tipos como estágio e TCC. Uma apresentação é realizada por alunos e ocorre perante uma banca. A avaliação do aluno, representada por um formulário de avaliação, é composta por itens que podem ter pesos associados, aos quais os avaliadores atribuem nota. Uma ata de

defesa/apresentação é emitida após a realização da banca. Uma banca é composta por avaliadores. Um dos avaliadores é o orientador do aluno, os demais avaliadores são professores ou profissionais. O trabalho apresentado pode ter coorientadores que podem ou não avaliar o trabalho. Os participantes da banca recebem comprovante de participação. Alunos que assistem à apresentação também recebem comprovante. Esse comprovante é uma listagem que contém o nome dos alunos presentes em determinada apresentação. Uma apresentação é realizada de acordo com uma agenda (dia e horário).

O Quadro 3 apresenta a listagem dos requisitos funcionais identificados para o sistema.

Identificação	Nome	Descrição
01	Fazer *upload* do trabalho a ser avaliado	O professor orientador ou o aluno faz *upload* do trabalho no sistema.
02	Fazer *upload* pelos membros da banca	Cada membro da banca pode fazer *upload* de trabalho com anotações ou de anotações em arquivo separado.
03	Registro de trabalho final "aprovado"	Os avaliadores de cada banca registram no sistema que a versão final do trabalho (pós-banca) atende as solicitações da banca e está aprovada para publicação, ou seja, como versão final.
04	Ata de defesa	Fazer *upload* da ata de defesa que deve ser anexada ao texto.
05	Postar ata digitalizada	O responsável pela atividade posta a ata digitalizada que fica anexada ao trabalho.
06	Avaliadores informar finalização de correções	Cada avaliador indica que as considerações foram atendidas, quando isso ocorrer.
07	Encerramento do projeto	O projeto é automaticamente finalizado pelo sistema após cada membro da banca indicar a finalização do trabalho e a banca realizar a avaliação.

Quadro 2 – Requisitos funcionais

No requisito fazer *upload* do trabalho é postado no sistema a versão do trabalho (monografia, relatório de estágio, proposta de TCC e outros). Esse arquivo é a versão que será disponibilizada para a banca e também a versão pós-banca para que os avaliadores possam verificar se as alterações solicitadas e sugeridas foram atendidas. O arquivo em versão para a banca e em versão pós-banca fica vinculado ao projeto que caracteriza o trabalho do aluno. O professor orientador faz o *upload* do arquivo, mas não pode editar os campos do formulário de cadastro do projeto. Isso é feito pelo responsável pela respectiva atividade.

O professor avaliador tem acesso aos trabalhos somente dos quais ele é banca. Para fazer *upload* de arquivo pode ser apresentado o formulário de cadastro de projeto.

A Figura 3 apresenta o processo para acompanhamento de avaliação do relatório de estágio e de trabalho de conclusão de curso.

Error! Objects cannot be created from editing field codes.

Figura 3 – Processo para acompanhamento de avaliação

Documento disponível para banca – após o orientador ou aluno fazer o *upload* do arquivo que será disponibilizado para a banca e a apresentação for agendada, o arquivo fica disponibilizado para a banca.

Banca informada do agendamento – todos os componentes da banca (orientador, coorientador e membros) ao acessarem o sistema visualizam os dados das bancas das quais eles fazem parte. E o arquivo com o trabalho fica disponível para *download*.

Avaliadores disponibilizam ajustes solicitados – após a realização da banca os membros avaliadores disponibilizam (por meio de *upload* no sistema) os ajustes solicitados no trabalho. Esses ajustes podem constar no próprio texto do aluno ou em outro arquivo.

Postada versão ajustada para banca verificar – após realizados os ajustes solicitados e/ou sugeridos pelos membros da banca uma nova versão é postada no sistema pelo professor orientador ou aluno e o texto fica disponível para acesso pelos componentes da banca.

Banca Informada da nova versão – a nova versão fica disponível no sistema para acesso aos membros da respectiva banca.

Versão final postada no sistema – quando a versão final é postada pelo orientador ou aluno o professor responsável pela atividade faz *upload* da ata de defesa digitalizada para ser anexada no texto, ficando disponível para *download*.

O diagrama de casos de uso apresentado na Figura 4 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: orientador, avaliador, responsável e administrador.

a) Aluno – o aluno faz a postagem (*upload*) do seu trabalho para avaliação, tem acesso (*download*) às correções indicadas e postadas pelos membros da banca e visualiza o resultado da avaliação do seu trabalho. O aluno também pode fazer *download* da ata de apresentação do seu trabalho para incluir no texto.

b) Orientador – o orientador faz a postagem (*upload*) do trabalho para avaliação, tem acesso (*download*) às correções indicadas e postadas pelos membros da banca e visualiza o resultado da avaliação dos seus orientados. O orientador também pode fazer *download* da ata de apresentação do trabalho para incluir no texto.

c) Avaliador – professores ou profissionais que compõem as bancas avaliando os trabalhos apresentados. Os avaliadores acessam o sistema para obter os trabalhos (*download*) que são membros das respectivas bancas e realizam a avaliação do trabalho. Os avaliadores podem postar no sistema (*upload*) observações (o próprio texto do trabalho editado ou um arquivo complementar). Esses arquivos ficam disponibilizados para acesso pelo aluno autor do trabalho e respectivo professor orientador. Os avaliadores têm acesso a *download* do trabalho a partir do momento que a banca é composta pelo professor responsável pela atividade e que o aluno ou orientador postar o trabalho.

d) Responsável – é o professor que é responsável pelo tipo de atividade que está sendo avaliada (estágio, trabalho de conclusão de curso ou outra) é quem realiza a atividade de compor bancas, agendar defesas, postar atas e registrar o resultado das bancas no sistema acadêmico. O professor responsável somente tem acesso ao tipo de trabalho ao qual ele é o responsável. A função de orientador é atribuída automaticamente para um professor que é membro de banca com o papel de orientador.

e) Administrador – é responsável pela manutenção de usuários e de tipos de atividades: estágio, TCC, propostas de TCC e outros. O administrador atribui esse tipo de atividade para um professor responsável que já está cadastrado no sistema como usuário.

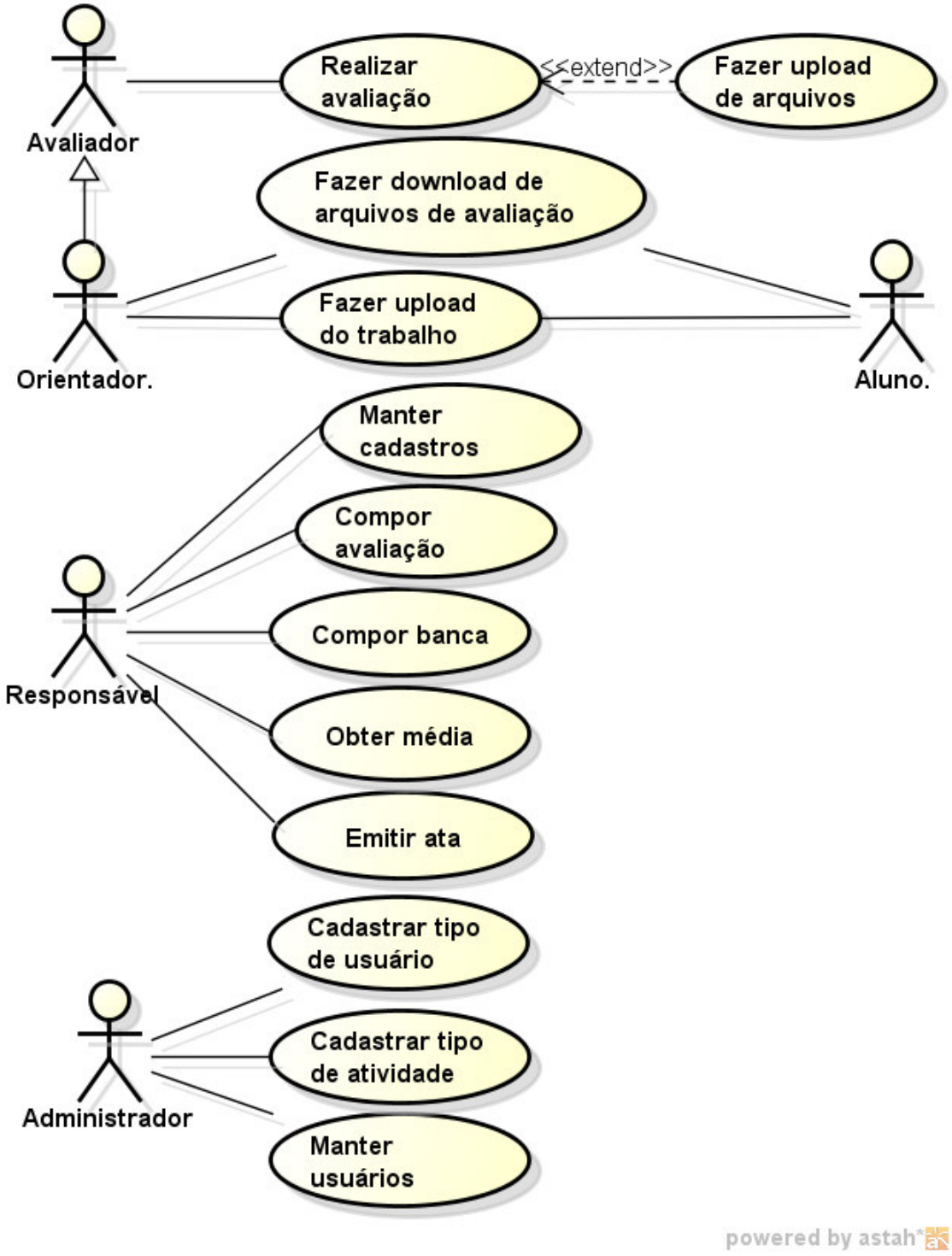

Figura 4 – Diagrama de casos de uso

Na Figura 5 está o diagrama de classes de análise do sistema. Nesse diagrama as classes modelo de avaliação, modelo de avaliação itens, item e grupo de item, permitem

compor um padrão (modelo) de avaliação que está vinculado a um curso e é utilizada para atribuir nota ao trabalho sendo apresentado. Uma apresentação ocorre de um projeto (estágio, trabalho de conclusão de curso, proposta de trabalho de conclusão de curso, entre outros) e está relacionada ao texto do trabalho do aluno (relatório) que é postado no sistema. Os avaliadores podem postar observações referentes ao projeto. E alunos são vinculados a projetos que são avaliados perante banca. Alunos e professores (no papel de avaliador, orientador, responsável pela atividade, profissionais) são atores com acesso ao sistema e, portanto, usuários.

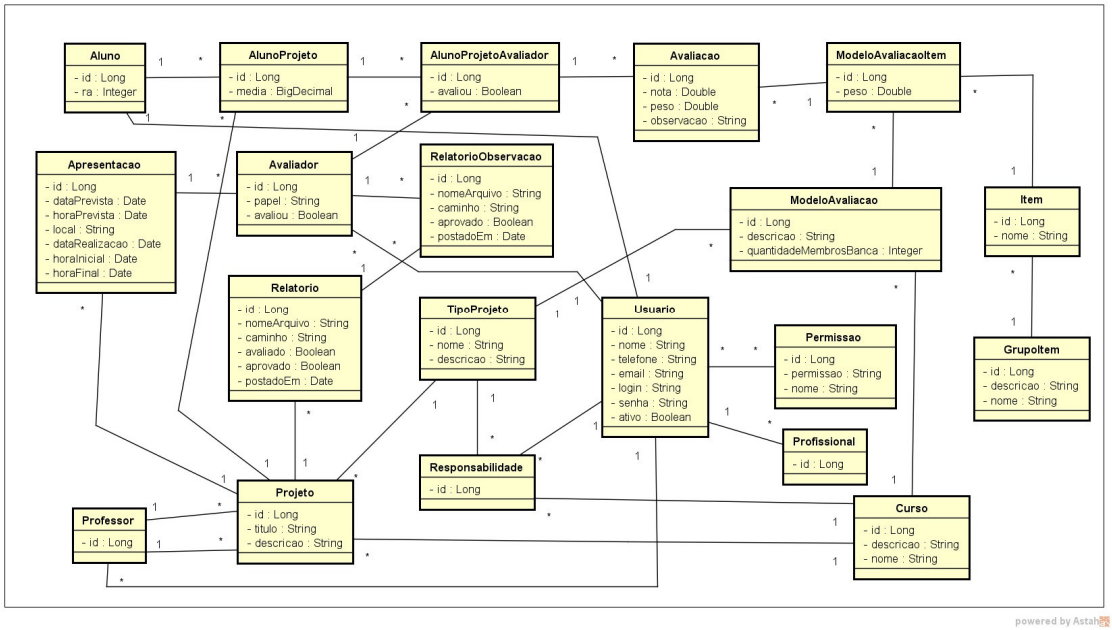

Figura 5 – Diagrama de classes de análise do sistema

A Figura 6 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação. As entidades do banco de dados são semelhantes às entidades do diagrama de classes (Figura 5). A tabela permissão armazena o acesso que cada tipo de usuário (perfil) possui no sistema.

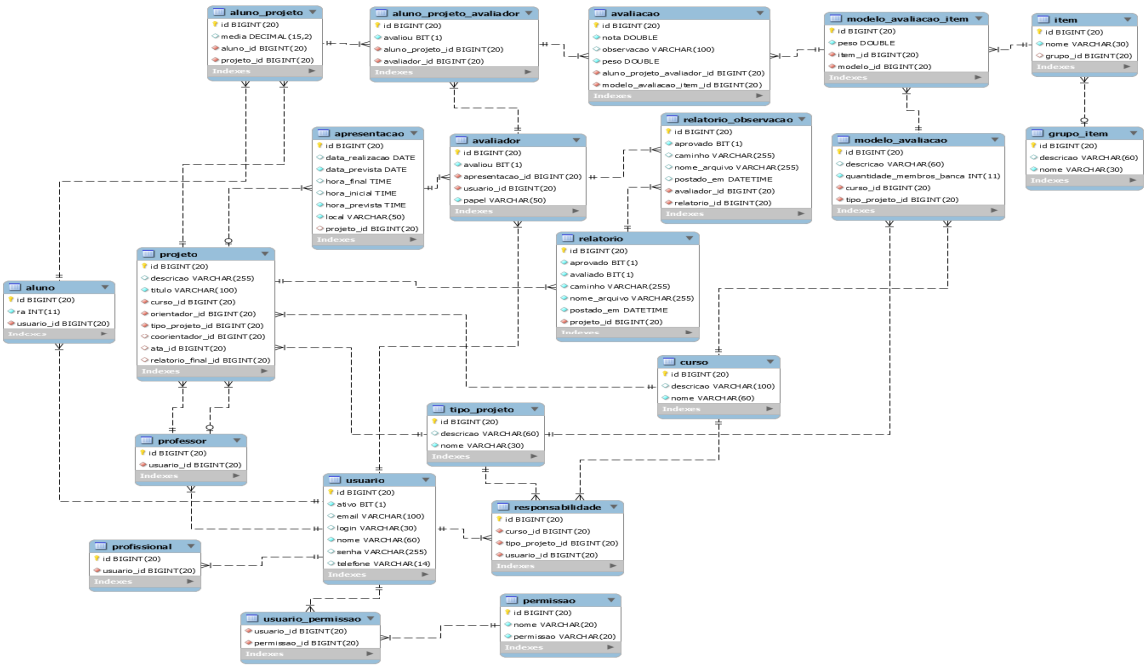

Figura 6 – Diagrama de entidades e relacionamentos do banco de dados

4.3 APRESENTAÇÃO DO SISTEMA

O leiaute do sistema é dividido em três partes: o cabeçalho, que ocupa o topo da página, o menu, que ocupa a lateral esquerda e o corpo que ocupa o restante da página. A Figura 7 apresenta o modelo de construção do leiaute.

Figura 7 – Leiaute do sistema

O cabeçalho é composto pela marca de identificação da UTFPR, que fica no lado esquerdo e um botão com o nome do usuário logado, que fica no lado direito. Ao clicar no botão expande-se um pequeno menu, com dois itens: o primeiro é o item “Sair”, ao clicar nesse item, o sistema efetua a ação de *logout* do usuário; e o segundo é o item “Meus Dados”, ao clicar nesse item, o sistema apresenta uma janela contendo os dados do usuário. A Figura 8 mostra o menu expandido.

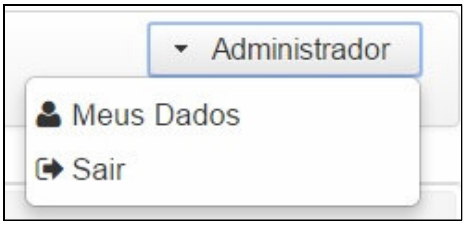

Figura 8 – Opções do usuário

A Figura 9 mostra a janela com os dados do usuário.

Figura 9 – Janela Meus Dados

Como visto na Figura 9, a janela contém os dados do usuário e duas opções. A primeira é a opção “Alterar Meus Dados”. Ao clicar nessa opção, o sistema apresenta um formulário com os campos que podem ser alterados pelo usuário. A segunda é a opção “Alterar Minha Senha”. Ao clicar nessa opção, o sistema apresenta outro formulário, contendo os campos necessários para a alteração da senha. A Figura 10 mostra o formulário para a alteração dos dados.

Figura 10 – Formulário para alteração dos dados

A Figura 11 mostra o formulário para a alteração da senha

Figura 11 – Formulário para alteração da senha

A Figura 12 mostra apenas o corpo da página, preenchido pelo conteúdo da tela de manutenção de cursos. Nessa página, o usuário consegue visualizar todos os cursos cadastrados e também realizar as operações básicas de cadastro como inclusão, alteração e exclusão.

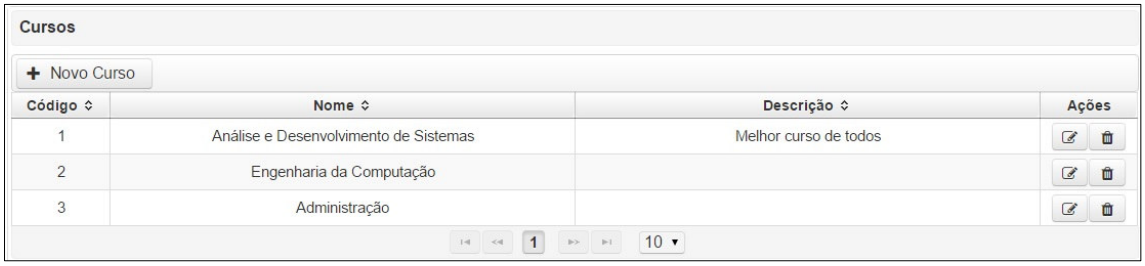

Figura 12 – Tela de manutenção de cursos

Ao clicar no botão “Novo Curso” (Figura 13) o sistema apresenta uma janela contendo um formulário com os campos necessários para a inclusão de um novo curso, como mostra a Figura 13.

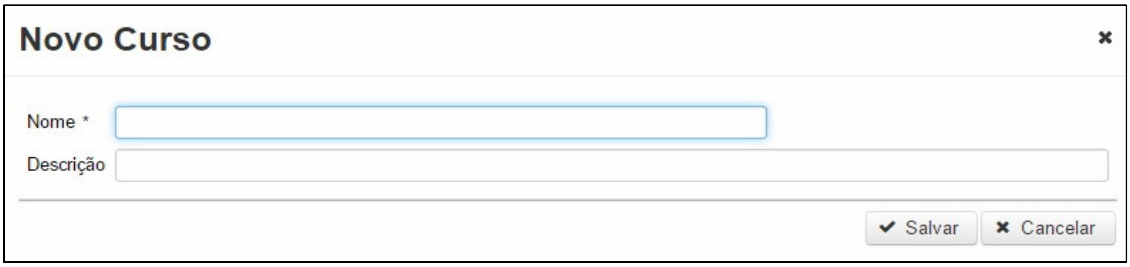

Figura 13 – Formulário para inclusão de um novo curso

Para incluir o novo curso, o usuário deve preencher os campos obrigatórios e clicar no botão “Salvar”.

A operação de alteração é semelhante à de inclusão, sendo que, para alterar algum curso o usuário deve clicar no botão “Alterar” da linha referente ao curso desejado. Dessa forma, o sistema fará o mesmo procedimento utilizado na inclusão, mas já trará o formulário preenchido com os dados do curso selecionado.

Para realizar a operação de exclusão, o usuário deve clicar no botão “Remover” da linha referente ao curso desejado. O sistema apresentará uma mensagem de confirmação, para não haver uma exclusão indesejada. Para prosseguir com a exclusão, o usuário deve clicar no botão “OK”.

As operações básicas como inclusão, alteração e exclusão foram apresentadas apenas com uma das entidades do sistema, a entidade de cursos. O processo para a manutenção das demais entidades é o mesmo.

Algumas telas do sistema possuem operações específicas, uma delas é a tela de manutenção de projetos. Essa tela pode ser vista de duas formas: se o usuário logado for um professor responsável, a tela será visualizada como mostra a Figura 14; se for um aluno ou avaliador, a tela será visualizada como mostra a Figura 15.

Figura 14 – Tela de manutenção de projetos para professor responsável

Figura 15 – Tela de manutenção de projetos para aluno ou avaliador

Como visto nas Figuras 14 e 15, apenas o professor responsável pode cadastrar, editar e excluir projetos, mas todos os usuários têm acesso à opção de visualização de postagens. Essa opção é específica da tela de manutenção de projetos e é nela que os usuários realizam a postagem da ata de defesa, relatórios para avaliação, observações para os relatórios e o relatório final. Mas, cada tipo de usuário está limitado a postar apenas o que é de sua responsabilidade. O professor responsável é o único que pode postar a ata de defesa, dessa forma, ele visualiza a tela de visualização de postagens como mostra a Figura 16.

Figura 16 – Tela de visualização de postagens para professor responsável

O aluno e o orientador do projeto, são os únicos que podem postar o relatório para avaliação e o relatório final, dessa forma, eles visualizam a tela de postagens como mostra a Figura 17.

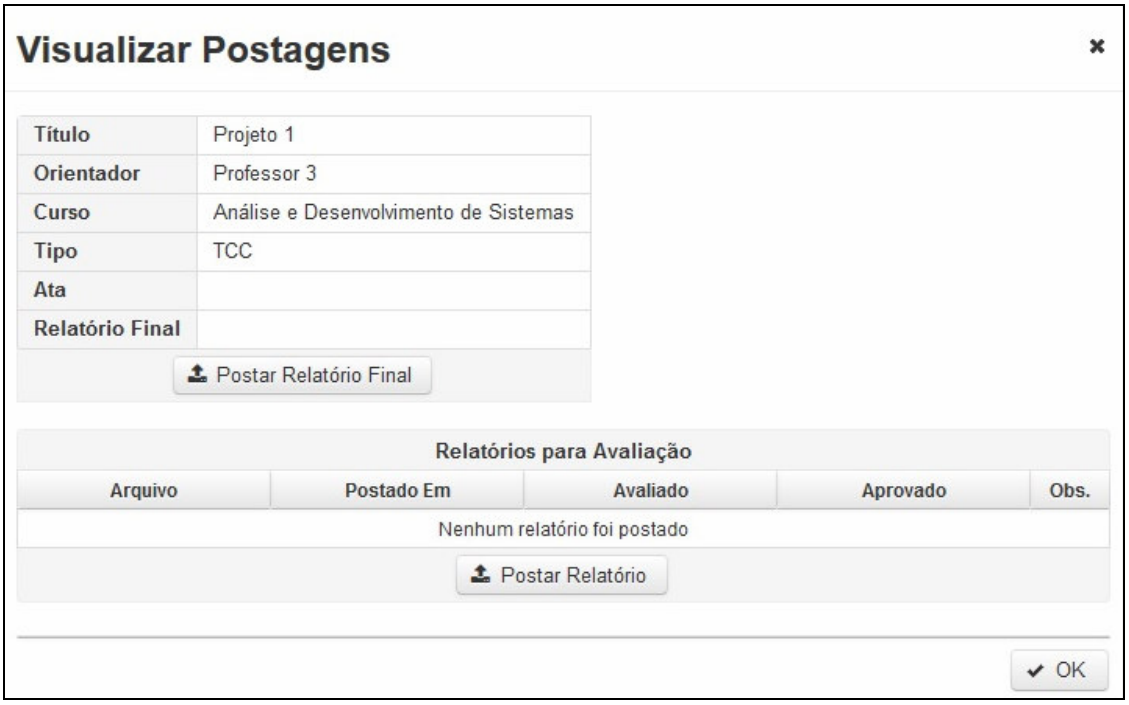

Figura 17 – Tela de visualização de postagens para aluno e orientador

Já os avaliadores, são os únicos que conseguem postar as observações em cada relatório para avaliação, dessa forma, eles visualizam a tela de postagens como mostra a Figura 18.

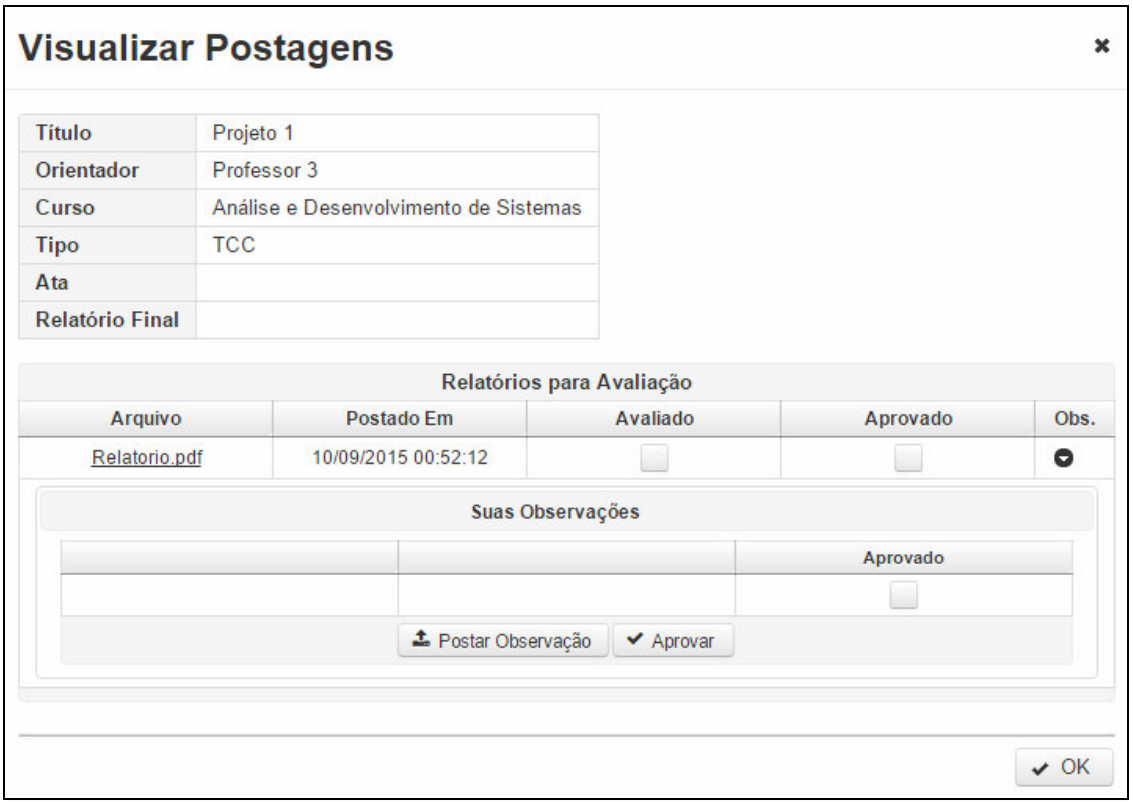

Figura 18 – Tela de visualização de postagens para avaliador

Após o projeto estar cadastrado, o usuário poderá cadastrar a apresentação do mesmo, através da tela de manutenção de avaliações. É no cadastro de apresentações que o usuário irá compor a banca avaliadora, a Figura 19 mostra o formulário de inclusão de uma apresentação.

Figura 19 – Formulário para inclusão de uma nova apresentação

Como visto na Figura 19, os campos "Data Realização", "Hora Inicial" e "Hora Final" não são obrigatórios, pois só serão preenchidos após a apresentação ter sido realizada. Após esses campos serem preenchidos, os membros da banca poderão avaliar a apresentação, através da tela de avaliações pendentes, como mostra a Figura 20.

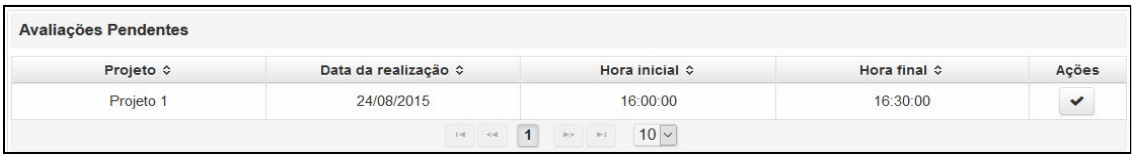

Figura 20 – Tela de avaliações pendentes

Como visto na Figura 20, o usuário visualizará todas as apresentações das quais ele faz parte e que ainda não foram avaliadas. Para realizar a avaliação, basta clicar no botão "Avaliar", dessa forma o sistema apresentará uma janela contendo algumas informações da apresentação e o formulário de avaliação, como mostra a Figura 21.

Figura 21 – Formulário de avaliação

Após todos os membros da banca terem realizado a avaliação de determinada apresentação, o sistema calculará a média final dos alunos. Os professores responsáveis podem acompanhar o andamento das avaliações através da tela de manutenção de apresentações, clicando no botão “Visualizar Avaliações”, após o usuário clicar nesse botão, o sistema apresentará uma janela que mostra algumas informações da apresentação, os alunos avaliados, a avaliação de cada um dos membros da banca e a média final do aluno, se já estiver calculada. A Figura 22 mostra a tela de visualização de avaliações.

Figura 22 – Visualização de avaliações

4.4 IMPLEMENTAÇÃO DO SISTEMA

O sistema foi implementado utilizando-se o *Integrated Develpment Environmet* (IDE) Eclipse Mars e o Maven, que vem instalado por padrão nessa versão da IDE. O Maven é uma ferramenta de gerenciamento, construção e implantação de projetos, criada pela Fundação Apache. Sua unidade básica de configuração é um arquivo XML chamado POM (*Project Object Model*) que deve ficar na raiz do projeto. Nesse arquivo, declara-se a estrutura, as dependências e as características do projeto. A Listagem 1 mostra o código do arquivo pom.xml.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>br.com.joaopichetti</groupId>
      <artifactId>tcc-especializacao</artifactId>
```

```
	<version>1.0</version>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.2.4.RELEASE</version>
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
		<start-class>br.com.joaopichetti.Application</start-class>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-tomcat</artifactId>
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>org.apache.tomcat.embed</groupId>
			<artifactId>tomcat-embed-jasper</artifactId>
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
		</dependency>
		<dependency>
			<groupId>com.sun.faces</groupId>
			<artifactId>jsf-api</artifactId>
			<version>2.2.10</version>
		</dependency>
		<dependency>
			<groupId>com.sun.faces</groupId>
			<artifactId>jsf-impl</artifactId>
			<version>2.2.10</version>
		</dependency>
		<dependency>
			<groupId>org.primefaces.extensions</groupId>
			<artifactId>primefaces-extensions</artifactId>
			<version>3.0.0</version>
		</dependency>
```

```
            <dependency>
                  <groupId>org.primefaces</groupId>
                  <artifactId>primefaces</artifactId>
                  <version>5.2</version>
            </dependency>
            <dependency>
                  <groupId>org.primefaces.themes</groupId>
                  <artifactId>bootstrap</artifactId>
                  <version>1.0.10</version>
            </dependency>
            <dependency>
                  <groupId>org.webjars</groupId>
                  <artifactId>font-awesome</artifactId>
                  <version>4.4.0</version>
            </dependency>
            <dependency>
                  <groupId>org.springframework.security</groupId>
                  <artifactId>spring-security-taglibs</artifactId>
            </dependency>
            <dependency>
                  <groupId>org.springframework.webflow</groupId>
                  <artifactId>spring-faces</artifactId>
                  <version>2.4.0.RELEASE</version>
            </dependency>
            <dependency>
                  <groupId>commons-fileupload</groupId>
                  <artifactId>commons-fileupload</artifactId>
                  <version>1.3.1</version>
            </dependency>
            <dependency>
                  <groupId>commons-io</groupId>
                  <artifactId>commons-io</artifactId>
                  <version>2.4</version>
            </dependency>
      </dependencies>

      <repositories>
            <repository>
                  <id>prime-repo</id>
                  <name>PrimeFaces Maven Repository</name>
                  <url>http://repository.primefaces.org</url>
                  <layout>default</layout>
            </repository>
      </repositories>
</project>
```

Listagem 1 – pom.xml

As dependências contidas no arquivo POM, são todas as bibliotecas externas que o projeto depende para ser construído. Essas dependências são gerenciadas pelo próprio Maven. Primeiramente, ele cria um diretório local e então, por meio da configuração contida no arquivo POM, baixa as dependências de um repositório *online* e as armazena no diretório criado. Ao ser feita a construção do projeto, o Maven busca as dependências necessárias, contidas no diretório local, e as adiciona ao projeto.

Além do gerenciamento das dependências, o Maven permite a criação do leiaute do projeto, ou seja, a estrutura de diretórios. Ele possui uma convenção para a criação, mas, também é possível criar uma estrutura customizada que é realizada por meio da configuração no arquivo POM. A Figura 23 mostra a estrutura do projeto, criada sem customização.

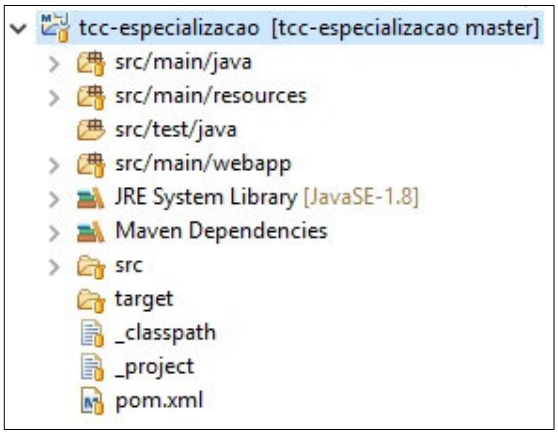

Figura 23 – Estrutura do projeto

Cada diretório da estrutura tem um propósito definido. O diretório "src/main/java" contém o código fonte da aplicação, ou seja, pacotes e arquivos Java. A Figura 24 mostra o conteúdo do diretório "src/main/java".

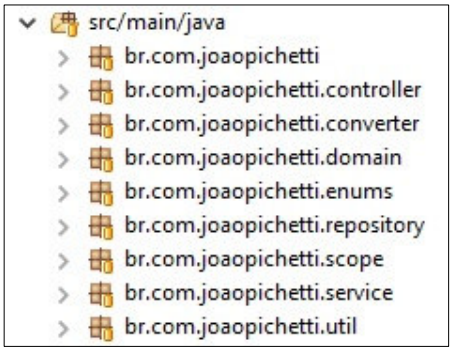

Figura 24 – Conteúdo do diretório "src/main/java"

Como visto na Listagem 1, uma das dependências utilizadas pelo projeto é o Spring Boot. Para iniciar a implementação da aplicação, basta criar uma classe que estenda uma classe do Spring, chamada "SprinBootServletInitializer", e implementar o método *main* (método principal, em aplicações Java) conforme mostra a Listagem 2.

```
package br.com.joaopichetti;

import java.util.HashMap;
import java.util.Map;
```

```
import javax.faces.webapp.FacesServlet;

import org.primefaces.webapp.filter.FileUploadFilter;
import org.springframework.beans.factory.config.CustomScopeConfigurer;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.embedded.FilterRegistrationBean;
import org.springframework.boot.context.embedded.ServletContextInitializer;
import
org.springframework.boot.context.embedded.ServletListenerRegistrationBean;
import org.springframework.boot.context.embedded.ServletRegistrationBean;
import org.springframework.boot.context.web.SpringBootServletInitializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.configuration.WebSecurit
yConfigurerAdapter;

import br.com.joaopichetti.scope.ViewScope;

import com.sun.faces.config.ConfigureListener;

@Configuration
@ComponentScan
@EnableAutoConfiguration
public class Application extends SpringBootServletInitializer {

	private static Class<Application> applicationClass =
Application.class;

	public static void main(String[] args) {
		SpringApplication.run(applicationClass, args);
	}

	@Override
	protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
		return application.sources(applicationClass);
	}

	@Bean
	public static ViewScope viewScope() {
		return new ViewScope();
	}

	@Bean
	public static CustomScopeConfigurer scopeConfigurer() {
		CustomScopeConfigurer configurer = new CustomScopeConfigurer();
		Map<String, Object> hashMap = new HashMap<>();
		hashMap.put("view", viewScope());
		configurer.setScopes(hashMap);
		return configurer;
	}

	@Bean
	public FacesServlet facesServlet() {
		return new FacesServlet();
	}
```

```
	@Bean
	public ServletRegistrationBean facesServletRegistration() {
		ServletRegistrationBean registration = new
ServletRegistrationBean(facesServlet(), "*.jsf");
		registration.setName("FacesServlet");
		return registration;
	}

	@Bean
	public ServletContextInitializer initializer() {
		return servletContext -> {
			servletContext.setInitParameter("primefaces.THEME",
"bootstrap");

	servletContext.setInitParameter("primefaces.FONT_AWESOME", "true");

	servletContext.setInitParameter("javax.faces.FACELETS_LIBRARIES",
"/WEB-INF/springsecurity.taglib.xml");
			servletContext.setInitParameter("primefaces.UPLOADER",
"commons");
		};
	}

	@Bean
	public ServletListenerRegistrationBean<ConfigureListener>
jsfConfigureListener() {
		return new ServletListenerRegistrationBean<>(new
ConfigureListener());
	}

	@Bean
	public WebSecurityConfigurerAdapter webSecurityConfigurerAdapter() {
		return new WebSecurityConfig();
	}

	@Bean
	public FilterRegistrationBean filterRegistrationBean() {
		FileUploadFilter fileUploadFilter = new FileUploadFilter();
		FilterRegistrationBean registrationBean = new
FilterRegistrationBean();
		registrationBean.setFilter(fileUploadFilter);

	registrationBean.addServletRegistrationBeans(facesServletRegistration
());
		return registrationBean;
	}

}
```

Listagem 2 – Application.java

A classe apresentada na Listagem 2 é a classe principal do sistema. Essa classe fica no pacote “br.com.joaopichetti”, é por meio dela que a aplicação é iniciada e as configurações do *framework* são realizadas. O Spring Boot contém o servidor Tomcat incorporado. Dessa forma, a aplicação não depende de um servidor de aplicações externo, bastando executar a

classe principal para que o Spring inicie o Tomcat e disponibilize a aplicação na porta configurada.

Como visto na Listagem 2, além do método *main*, existem vários métodos, anotados com a anotação "Bean". Essa anotação é utilizada para que o Spring compreenda como determinados objetos devem ser instanciados. Nesse caso, os métodos estão sendo utilizados para alterar a configuração padrão do Spring. Uma das configurações, realizada pelo método "webSecurityConfigurerAdapter", refere-se ao módulo de segurança. Esse método retorna um novo objeto da classe "WebSecurityConfig", apresentada na Listagem 3.

```
package br.com.joaopichetti;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.authentication.builders.Auth
enticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurit
yConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

import br.com.joaopichetti.repository.UsuarioRepository;
import br.com.joaopichetti.service.UsuarioService;

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

	@Autowired
	private UsuarioRepository usuarioRepository;

	@Override
	protected void configure(AuthenticationManagerBuilder auth)
				throws Exception {
		auth.userDetailsService(userDetailsService())
			.passwordEncoder(passwordEncoder());
	}

	@Bean
	public PasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder(10);
	}

	@Bean
	public UserDetailsService userDetailsService() {
		return new UsuarioService(usuarioRepository);
	}

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.csrf().disable().authorizeRequests()
			.antMatchers("/resources/**").permitAll()

	.antMatchers("/admin/aluno**").hasAnyRole("ADMIN","RESPONSAVEL")
```

```
        .antMatchers("/admin/apresentacao**").hasRole("RESPONSAVEL")

        .antMatchers("/admin/avaliacao**").hasAnyRole("AVALIADOR","RESPONSAVE
L")
                .antMatchers("/admin/curso**").hasRole("ADMIN")
                .antMatchers("/admin/grupoitem**").hasRole("RESPONSAVEL")
                .antMatchers("/admin/item**").hasRole("RESPONSAVEL")
                .antMatchers("/admin/modeloavaliacao**").hasRole("RESPONSAVEL")
                .antMatchers("/admin/papel**").hasRole("RESPONSAVEL")

.antMatchers("/admin/professor**").hasAnyRole("ADMIN","RESPONSAVEL")

.antMatchers("/admin/profissional**").hasAnyRole("ADMIN","RESPONSAVEL")

.antMatchers("/admin/projeto**").hasAnyRole("RESPONSAVEL","ALUNO","AVALIADO
R")
                .antMatchers("/admin/tipoprojeto**").hasRole("ADMIN")
                .antMatchers("/admin/usuario**").hasRole("ADMIN")

        .antMatchers("/admin/**").hasAnyRole("ADMIN","RESPONSAVEL","AVALIADOR
","ALUNO")
                        .and()
                        .formLogin().loginPage("/login").defaultSuccessUrl("/")
                                .loginProcessingUrl("/j_spring_security_check")
                                .failureUrl("/login").permitAll()
                        .and()
                        .logout().logoutSuccessUrl("/login").permitAll();
        }

}
```

Listagem 3 – WebSecurityConfig.java

Como visto na Listagem 3, a classe "WebSecurityConfig" estende uma classe do Spring chamada "WebSecurityConfigurerAdapter". Essa classe está contida no módulo Spring Security. Ainda na Listagem 3, há um método chamado "configure" que recebe um objeto da classe "HttpSecurity" por parâmetro. É nesse método que é feita a configuração do controle de acesso das URLs, permitindo ou negando acesso de acordo com a URL digitada e do perfil do usuário logado.

Outro módulo utilizado no projeto é o Spring Data JPA. O JPA (*Java Persistence API*) é uma especificação Java, o *Java Specification Requests* (JSR 317) que deve ser seguida por *frameworks* de mapeamento objeto-relacional (*Object-Relational Mapping* (ORM)), como o Hibernate. O Spring Data JPA não é um *framework* ORM, mas foi desenvolvido com base no padrão JPA 2, para trabalhar com qualquer *framework* que siga a especificação.

O pacote "br.com.joaopichetti.domain" contém todas as classes de entidade, ou seja, as classes que fazem referência às tabelas do banco de dados. Essas classes utilizam anotações do JPA para que seja feito o mapeamento objeto-relacional. A Listagem 4 apresenta o código da classe "BaseDomain", utilizada como base para todas as classes de entidade. Essa classe

contém um método abstrato chamado “getId” que retorna um valor do tipo “Long”. Dessa forma, as classes filhas devem possuir um atributo chamado “id” que seja do tipo “Long”. Esse atributo faz referência à chave primária utilizada na tabela cuja classe faz referência. Além disso, a classe “BaseDomain” sobrescreve os métodos “equals” e “hashCode”, presentes na superclasse “Object”, para que seja utilizado apenas o método “getId” na comparação entre objetos.

```
package br.com.joaopichetti.domain;
import java.io.Serializable;

public abstract class BaseDomain implements Serializable {
	private static final long serialVersionUID = 4325011599295320476L;
	protected abstract Long getId();
	@Override
	public boolean equals(Object o) {
		if (this == o)
			return true;
		if (o == null || getClass() != o.getClass())
			return false;
		BaseDomain that = (BaseDomain) o;
		return !(getId() != null ? !getId().equals(that.getId()) :
that.getId() != null);
	}
	@Override
	public int hashCode() {
		return getId() != null ? getId().hashCode() : 0;
	}
}
```

Listagem 4 – BaseDomain.java

A Listagem 5 apresenta o código da classe “Curso”, que é uma subclasse de “BaseDomain” e que, como citado anteriormente, possui as anotações do JPA e o atributo “id” do tipo “Long”.

```
package br.com.joaopichetti.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Curso extends BaseDomain {
	private static final long serialVersionUID = 5394397052652304848L;
	@Id
	@GeneratedValue
	private Long id;
	@Column(length=100)
	private String descricao;
	@Column(length=60, nullable=false)
	private String nome;

	public Curso() {}
```

```
	public Curso(Long id) {
		this.id = id;
	}

	public Curso(Long id, String nome) {
		this(id);
		this.nome = nome;
	}

	public Long getId() {
		return id;
	}

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

	public String getDescricao() {
		return this.descricao;
	}

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

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

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

Listagem 5 – Curso.java

O Spring Data JPA é responsável pela implementação da camada de persistência de dados, utilizando repositórios, não sendo necessário criar classes concretas de acesso ao banco de dados, conhecidas como DAOs (*Data Access Objects*), bastando apenas criar uma interface Java para cada classe de entidade e estender uma interface chamada "JpaRepository". Essa interface possui todos os métodos referentes às operações *Create-Read-Update-Delete* (CRUD), além de métodos para paginação de dados. Além disso, ao ser realizada a herança, o Spring reconhece a interface criada como um *bean*. Dessa forma ele consegue gerenciar a injeção de dependência sobre a mesma.

O pacote "br.com.joaopichetti.repository" contém todos os repositórios e a listagem 6 apresenta o código da interface "CursoRepository".

```
package br.com.joaopichetti.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import br.com.joaopichetti.domain.Curso;

public interface CursoRepository extends JpaRepository<Curso, Long> {

	final String FIND_ALL_TO_LOOKUP = "SELECT new Curso(c.id, c.nome)
FROM Curso c";

	@Query(FIND_ALL_TO_LOOKUP)
	List<Curso> findAllToLookup();

}
```

Listagem 6 – CursoRepository.java

Como visto na Listagem 6, a interface "CursoRepository" estende a interface "JpaRepository". Essa interface utiliza o recurso *Generics* do Java exigindo a passagem de dois tipos de dados após a declaração da mesma. O primeiro tipo refere-se à classe de entidade e o segundo refere-se ao tipo da chave primária contida na classe de entidade. Dessa forma o Spring reconhece em qual tabela do banco de dados as operações devem ser realizadas. Se houver a necessidade de criar um método customizado, basta declarar a assinatura do mesmo e anotá-lo com a anotação "Query", passando por parâmetro um comando em linguagem *Java Persistence Query Language* (JPQL), como é o caso do método "findAllToLookup" contido na interface "CursoRepository".

Outro módulo utilizado é o Spring Web, esse módulo serve para o desenvolvimento de aplicações Web MVC, como a aplicação também utiliza o JSF, o controlador faz o papel de ManagedBean. O pacote "br.com.joaopichetti.controller" contém todos os controladores utilizados na aplicação. A Listagem 7 apresenta o código da classe "CrudBaseController".

```
package br.com.joaopichetti.controller;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import br.com.joaopichetti.domain.BaseDomain;
import br.com.joaopichetti.service.FacesService;
import br.com.joaopichetti.service.UsuarioLogadoService;

public abstract class CrudBaseController<D extends BaseDomain, R extends
JpaRepository<D, Long>> implements Serializable {

	private static final long serialVersionUID = -8990751501254467804L;
```

```
	private D instancia;
	private Class<D> domainClass;
	private R domainRepository;
	private List<D> lista = new ArrayList<>();
	private FacesService faces;
	private UsuarioLogadoService usuarioLogado;
	private String descricaoForm;
	private List<Coluna> colunas = new ArrayList<>();

	// CONSTRUTORES
	protected CrudBaseController(Class<D> domainClass, R
domainRepository,
			FacesService faces) {
		this(domainClass, domainRepository, faces, null);
	}

	protected CrudBaseController(Class<D> domainClass, R
domainRepository,
			FacesService faces, UsuarioLogadoService usuarioLogado) {

		this.domainClass = domainClass;
		this.domainRepository = domainRepository;
		this.faces = faces;
		this.usuarioLogado = usuarioLogado;
		criarColunaId();
		criarColunas();
		carregarLista();
	}

	// METODOS
	protected void addColuna(Coluna coluna) {
		colunas.add(coluna);
	}

	public void cancelar() {
		instancia = null;
		faces.hideDialog("dlgForm");
	}

	protected void carregarDetalhes() {
		// sobreescrever nos filhos, se necessario
	}

	protected void carregarLista() {
		lista = domainRepository.findAll();
	}

	protected void carregarLookups() {
		// sobreescrever nos filhos, se necessario
	}

	private void criarColunaId() {
		colunas = new ArrayList<>();
		addColuna(new Coluna().setCampo("id").setCodDesc("comum.id")
				.setLargura("75"));
	}

	protected abstract void criarColunas();

	public boolean dadosValidos() {
```

```
            return true;
      }

      public void editar(Long id) {
            D instancia = domainRepository.findOne(id);
            if (instancia == null) {
                  faces.addMensagemErro(getDomainName() + ".inexistente");
                  carregarLista();
            } else {
                  this.instancia = instancia;
                  carregarLookups();
                  carregarDetalhes();
                  descricaoForm = getDomainName() + ".editar";
                  faces.showDialog("dlgForm");
            }
      }

      public void excluir(Long id) {
            try {
                  domainRepository.delete(id);
                  faces.addMensagemSucesso(getDomainName() +
".excluir.ok");
            } catch (Exception ex) {
                  faces.addMensagemErro(getDomainName() + ".excluir.erro");
                  ex.printStackTrace();
            } finally {
                  carregarLista();
            }
      }

      public D getNovaInstancia() {
            try {
                  return domainClass.newInstance();
            } catch (InstantiationException | IllegalAccessException e) {
                  e.printStackTrace();
                  return null;
            }
      }

      public void inserir() {
            instancia = getNovaInstancia();
            carregarLookups();
            descricaoForm = getDomainName() + ".inserir";
            faces.showDialog("dlgForm");
      }

      public void salvar() {
            if (dadosValidos()) {
                  try {
                        domainRepository.save(instancia);
                        faces.addMensagemSucesso(getDomainName() +
".salvar.ok");
                  } catch (Exception ex) {
                        faces.addMensagemErro(getDomainName() +
".salvar.erro");
                        ex.printStackTrace();
                  } finally {
                        carregarLista();
                        faces.hideDialog("dlgForm");
                  }
            }
```

```
	}

	// GETTERS & SETTERS
	public D getInstancia() {
		return instancia;
	}

	public void setInstancia(D instancia) {
		this.instancia = instancia;
	}

	public List<D> getLista() {
		return lista;
	}

	public void setLista(List<D> lista) {
		this.lista = lista;
	}

	public String getDescricaoForm() {
		return descricaoForm;
	}

	public void setDescricaoForm(String descricaoForm) {
		this.descricaoForm = descricaoForm;
	}

	public List<Coluna> getColunas() {
		return colunas;
	}

	public void setColunas(List<Coluna> colunas) {
		this.colunas = colunas;
	}

	public String getDomainName() {
		return domainClass.getSimpleName().toLowerCase();
	}

	public R getDomainRepository() {
		return domainRepository;
	}

	protected FacesService getFaces() {
		return faces;
	}

	protected UsuarioLogadoService getUsuarioLogado() {
		return usuarioLogado;
	}

}
```

Listagem 7 – CrudBaseController.java

A classe “CrudBaseController” (Listagem 7), é uma classe abstrata que utiliza o recurso *Generics*, exigindo a passagem de dois tipos de dados após sua declaração. O primeiro tipo refere-se à classe de entidade e o segundo refere-se ao repositório referente à

classe de entidade. Essa classe implementa os métodos de inserção, edição, exclusão e listagem. Dessa forma, se necessário, as classes filhas apenas sobrescrevem esses métodos. A Listagem 8 apresenta o código da classe “CursoController”.

```
package br.com.joaopichetti.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

import br.com.joaopichetti.domain.Curso;
import br.com.joaopichetti.repository.CursoRepository;
import br.com.joaopichetti.service.FacesService;

@Controller
@Scope("view")
public class CursoController extends CrudBaseController<Curso,
CursoRepository> {

	private static final long serialVersionUID = 5765367490327153274L;

	@Autowired
	protected CursoController(CursoRepository domainRepository,
FacesService faces) {
		super(Curso.class, domainRepository, faces);
	}

	@Override
	protected void criarColunas() {
		super.addColuna(new
Coluna().setCampo("nome").setCodDesc("comum.nome"));
		super.addColuna(new
Coluna().setCampo("descricao").setCodDesc("comum.descricao"));
	}

}
```

Listagem 8 – CursoController.java

Como visto na Listagem 8, a classe “CursoController” estende a classe “CrudBaseController” (Listagem 7). Além disso, ela possui duas anotações, a anotação “Controller” serve para que o Spring reconheça a classe como um controlador e a anotação “Scope” “mostra” ao Spring qual é o escopo do controlador, nesse caso, o escopo é *view* (visão) que mantém o controlador instanciado enquanto a página estiver disponível. Além disso, a classe sobrescreve o método “criarColunas”. Esse método serve para criar uma lista com objetos da classe “Coluna”, essa lista é utilizada para a criação dinâmica das colunas da tabela de listagem pelo PrimeFaces. A Listagem 9 apresenta o código da classe “Coluna”.

```java
package br.com.joaopichetti.controller;

import org.apache.commons.lang3.StringUtils;

public class Coluna {

    private String campo;
    private String campoSort;
    private String codDesc;
    private String largura;

    public Coluna() {
      campo = "";
      campoSort = "";
      codDesc = "";
      largura = "";
    }

      public String getCampo() {
            return campo;
      }

      public Coluna setCampo(String campo) {
            this.campo = campo;
            return this;
      }

      public String getCampoSort() {
            return StringUtils.isBlank(campoSort) ? campo : campoSort;
      }

      public Coluna setCampoSort(String campoSort) {
            this.campoSort = campoSort;
            return this;
      }

      public String getCodDesc() {
            return codDesc;
      }

      public Coluna setCodDesc(String codDesc) {
            this.codDesc = codDesc;
            return this;
      }

      public String getLargura() {
            return largura;
      }

      public Coluna setLargura(String largura) {
            this.largura = largura;
            return this;
      }

}
```

Listagem 9 – Coluna.java

Como citado anteriormente, o escopo da classe “CursoController” é de visão. Esse escopo é nativo em aplicações JSF, mas nesse caso, a aplicação é gerenciada pelo Spring que

não possui o escopo de visão, mas em compensação ele permite a criação de escopos customizados. Dessa forma é possível implementar o escopo de visão. São necessárias duas classes para a implementação desse escopo, essas classes são apresentadas na Listagens 10 e 11, respectivamente. Essas classes ficam no pacote “br.com.joaopichetti.scope”.

```
package br.com.joaopichetti.scope;

import java.util.HashMap;
import java.util.Map;

import javax.faces.context.FacesContext;

import org.springframework.beans.factory.ObjectFactory;
import org.springframework.beans.factory.config.Scope;
import org.springframework.web.context.request.FacesRequestAttributes;

public class ViewScope implements Scope {

	public static final String VIEW_SCOPE_CALLBACKS =
"viewScope.callbacks";

	@Override
	public synchronized Object get(String name, ObjectFactory<?>
objectFactory) {
		Object instance = getViewMap().get(name);
		if (instance == null) {
			instance = objectFactory.getObject();
			getViewMap().put(name, instance);
		}
		return instance;
	}

	@Override
	public Object remove(String name) {
		Object instance = getViewMap().remove(name);
		if (instance != null) {
			@SuppressWarnings("unchecked")
			Map<String, Runnable> callbacks = (Map<String, Runnable>)
getViewMap().get(VIEW_SCOPE_CALLBACKS);
			if (callbacks != null) {
				callbacks.remove(name);
			}
		}
		return instance;
	}

	@Override
	public void registerDestructionCallback(String name, Runnable
runnable) {
		@SuppressWarnings("unchecked")
		Map<String, Runnable> callbacks = (Map<String, Runnable>)
getViewMap().get(VIEW_SCOPE_CALLBACKS);
		if (callbacks != null) {
			callbacks.put(name, runnable);
		}
	}

	@Override
	public Object resolveContextualObject(String name) {
```

```
            FacesContext facesContext = FacesContext.getCurrentInstance();
            FacesRequestAttributes facesRequestAttributes = new
FacesRequestAttributes(facesContext);
            return facesRequestAttributes.resolveReference(name);
      }

      @Override
      public String getConversationId() {
            FacesContext facesContext = FacesContext.getCurrentInstance();
            FacesRequestAttributes facesRequestAttributes = new
FacesRequestAttributes(facesContext);
            return facesRequestAttributes.getSessionId() + "-" +
facesContext.getViewRoot().getViewId();
      }

      private Map<String, Object> getViewMap() {
            if (FacesContext.getCurrentInstance() != null &&
FacesContext.getCurrentInstance().getViewRoot() != null
                        &&
FacesContext.getCurrentInstance().getViewRoot().getViewMap() != null) {
                  return
FacesContext.getCurrentInstance().getViewRoot().getViewMap();
            } else {
                  return new HashMap<String, Object>();
            }
      }

}
```

Listagem 10 – ViewScope.java

```
package br.com.joaopichetti.scope;

import java.util.HashMap;
import java.util.Map;

import javax.faces.component.UIViewRoot;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.PostConstructViewMapEvent;
import javax.faces.event.PreDestroyViewMapEvent;
import javax.faces.event.SystemEvent;
import javax.faces.event.ViewMapListener;

public class ViewScopeCallbackRegistrer implements ViewMapListener {

      @Override
      public void processEvent(SystemEvent event) throws
AbortProcessingException {
            if (event instanceof PostConstructViewMapEvent) {
                  PostConstructViewMapEvent viewMapEvent =
(PostConstructViewMapEvent) event;
                  UIViewRoot viewRoot = (UIViewRoot)
viewMapEvent.getComponent();
                  viewRoot.getViewMap().put(ViewScope.VIEW_SCOPE_CALLBACKS,
new HashMap<String, Runnable>());
            } else if (event instanceof PreDestroyViewMapEvent) {
                  PreDestroyViewMapEvent viewMapEvent =
(PreDestroyViewMapEvent) event;
                  UIViewRoot viewRoot = (UIViewRoot)
viewMapEvent.getComponent();
                  @SuppressWarnings("unchecked")
```

```
                    Map<String, Runnable> callbacks = (Map<String, Runnable>)
viewRoot.getViewMap().get(ViewScope.VIEW_SCOPE_CALLBACKS);
                    if (callbacks != null) {
                          for (Runnable c : callbacks.values()) {
                                c.run();
                          }
                          callbacks.clear();
                    }
              }
        }

        @Override
        public boolean isListenerForSource(Object source) {
              return source instanceof UIViewRoot;
        }

}
```

Listagem 11 – ViewScopeCallbackRegister.java

Outro diretório da estrutura é o "src/main/resources", esse diretório contém apenas dois arquivos, como mostra a Figura 25.

Figura 25 – Diretório "src/main/resources"

O arquivo "application.properties" (Listagem 12), contém algumas propriedades responsáveis por definir a conexão com o banco de dados, como a URL de conexão, o usuário e a senha.

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.url=jdbc:mysql://localhost:3306/tcc_especializacao
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driverClassName=com.mysql.jdbc.Driver
server.port=8080
```

Listagem 12 – application.properties

Já o arquivo "messages.properties" (Listagem 13), é utilizado para realizar a internacionalização da aplicação, dessa forma, nas telas do sistema, ao invés do texto, utiliza-se o código para internacionalização e é no arquivo de internacionalização que introduz-se o texto.

```
#_COMUM
comum.acoes=Ações
comum.ok=OK
comum.cancelar=Cancelar
```

```
comum.salvar=Salvar
comum.atencao=Atenção
comum.avaliar=Avaliar
comum.editar=Editar
comum.excluir=Excluir
comum.mostrar=Mostrar
comum.ativos=Ativos
comum.inativos=Inativos
comum.todos=Todos
comum.selecionar=Selecionar
comum.enviar=Enviar
comum.visualizarComo=Visualizar como
comum.fechar=Fechar

#_COMUM_CAMPOS
comum.id=Código
comum.nome=Nome
comum.descricao=Descrição
comum.usuario=Usuário
comum.senha=Senha
comum.curso=Curso
comum.tipoProjeto=Tipo de Projeto
comum.responsavel=Responsável
comum.avaliador=Avaliador

#_COMUM_CAMPOS_VALIDACAO
comum.nome.required=Informe o nome
comum.usuario.required=Informe o usuário
comum.senha.required=Informe a senha
comum.curso.required=Selecione o curso
comum.tipoProjeto.required=Selecione o tipo de projeto

#ALUNO
aluno.listagem.descricao=Alunos
aluno.listagem.tabela.vazia=Nenhum aluno cadastrado
aluno.inserir=Novo Aluno
aluno.editar=Editar Aluno
aluno.excluir.confirmar=O aluno será excluído definitivamente
aluno.excluir.ok=O aluno foi excluído com sucesso!
aluno.excluir.erro=Não foi possível excluír o aluno
aluno.salvar.ok=O aluno foi cadastrado com sucesso!
aluno.salvar.erro=Não foi possível cadastrar o aluno
aluno.inexistente=Esse aluno foi alterado ou removido por outro usuário

#ALUNO_CAMPOS
aluno.ra=R.A.

#ALUNO_CAMPOS_VALIDACAO
aluno.ra.required=Informe o R.A.
aluno.ra.validateLongRange=O R.A. deve ser maior que zero
...
```

Listagem 13 – messages.properties

Outro diretório é o “src/main/webapp”, é nesse diretório que ficam os arquivos referentes a parte do lado cliente, como as páginas, arquivos de estilo e imagens. A Figura 26 mostra o conteúdo do diretório “src/main/webapp”.

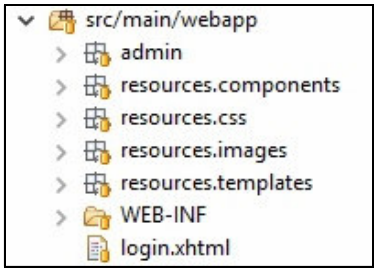

Figura 26 – Diretório "src/main/webapp"

A aplicação utiliza o JSF que fornece um subprojeto chamado "Facelets", esse projeto facilita a criação de *templates*. A Listagem 14 mostra o código do arquivo "base.xhtml", utilizado como base para a criação do leiaute.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:sec="http://www.springframework.org/security/tags">

<h:head>
    <title>Sistema</title>
    <link href="/resources/css/geral.css" rel="stylesheet" />
</h:head>

<h:body>
    <p:layout fullPage="true">
        <p:layoutUnit position="north" styleClass="sem-borda">
            <ui:include src="cabecalho.xhtml" />
        </p:layoutUnit>
        <p:layoutUnit header="Menu" position="west" size="250" collapsible="true">
            <ui:include src="menu.xhtml"/>
        </p:layoutUnit>
        <p:layoutUnit position="center">
              <p:growl id="messages" showDetail="true" showSummary="false"
                       autoUpdate="true"/>
              <p:confirmDialog global="true" showEffect="clip"
hideEffect="explode">
                    <div align="right">
                          <p:commandButton value="#{msg['comum.ok']}"
type="button"
                          styleClass="ui-confirmdialog-yes" icon="fa fa-check"/>

                       <p:commandButton value="#{msg['comum.cancelar']}"
type="button"
                          styleClass="ui-confirmdialog-no" icon="fa fa-close"/>
                 </div>
              </p:confirmDialog>
              <ui:insert name="conteudo">
                  <!-- Aqui ficará o conteúdo preenchido pelas outras páginas -->
              </ui:insert>
        </p:layoutUnit>
        <p:ajaxStatus style="position: absolute; right: 3%; top: 90%;"
```

```
                    id="ajaxStatusPanel">
            <f:facet name="start">
                <h:graphicImage value="/resources/images/loading.gif"/>
            </f:facet>

            <f:facet name="complete">
                <h:outputText value=""/>
            </f:facet>
        </p:ajaxStatus>
        …
    </p:layout>
</h:body>
</html>
```

Listagem 14 – base.xhtml

Ainda há dois arquivos utilizados na criação do leiaute: um deles é o arquivo "cabecalho.xhtml" (Listagem 15) e o outro é o arquivo "menu.xhtml" (Listagem 16). Os arquivos do leiaute ficam no diretório "resources/templates".

```
<?xml version="1.0" encoding="UTF-8" ?>
<ui:fragment xmlns:ui="http://java.sun.com/jsf/facelets"
             xmlns:h="http://java.sun.com/jsf/html"
             xmlns:p="http://primefaces.org/ui"
             xmlns:f="http://java.sun.com/jsf/core">
    <h:form>
          <p:menubar>
            <p:menuitem>
                  <p:commandLink action="/admin/index?faces-redirect=true"
title="Home">
                        <h:graphicImage
value="/resources/images/logo.png" width="81" height="30"
                              style="margin: 10px;" />
                    </p:commandLink>
            </p:menuitem>
              <f:facet name="options">
                  <p:menuButton
value="#{loginController.usuarioLogado.nome}">
                        <p:menuitem value="Meus Dados" icon="fa fa-user"
      action="#{loginController.mostrarDialogDadosUsuario}"
                              update=":#{p:component('dlgMeusDados')}" />
                        <p:menuitem value="Sair" icon="fa fa-sign-out"
                              onclick="location.href='/logout'" />
                  </p:menuButton>
              </f:facet>
          </p:menubar>
      </h:form>
</ui:fragment>
```

Listagem 15 – cabecalho.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:fragment xmlns="http://www.w3.org/1999/xhtml"
             xmlns:h="http://java.sun.com/jsf/html"
             xmlns:ui="http://java.sun.com/jsf/facelets"
             xmlns:p="http://primefaces.org/ui"
             xmlns:sec="http://www.springframework.org/security/tags">
```

```
    <h:form>
        <p:panelMenu>
            <p:submenu label="Menu">
                  <p:menuitem value="#{msg['aluno.listagem.descricao']}"
                        action="/admin/aluno?faces-redirect=true"
      rendered="#{sec:areAnyGranted('ROLE_ADMIN,ROLE_RESPONSAVEL')}" />
                  <p:menuitem
value="#{msg['apresentacao.listagem.descricao']}"
                  action="/admin/apresentacao?faces-redirect=true"
                     rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL')}" />
                  <p:menuitem
value="#{msg['avaliacaopendente.listagem.descricao']}"
                  action="/admin/avaliacao?faces-redirect=true"
                        rendered="#{sec:areAnyGranted('ROLE_AVALIADOR,
ROLE_RESPONSAVEL')}" />
                <p:menuitem value="#{msg['curso.listagem.descricao']}"
                  action="/admin/curso?faces-redirect=true"
                        rendered="#{sec:areAnyGranted('ROLE_ADMIN')}" />
                <p:menuitem value="#{msg['grupoitem.listagem.descricao']}"
                  action="/admin/grupoitem?faces-redirect=true"
                  rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL')}" />
                <p:menuitem value="#{msg['item.listagem.descricao']}"
                  action="/admin/item?faces-redirect=true"
                   rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL')}" />
                <p:menuitem
value="#{msg['modeloavaliacao.listagem.descricao']}"
                  action="/admin/modeloavaliacao?faces-redirect=true"
                  rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL')}" />
                <p:menuitem value="#{msg['papel.listagem.descricao']}"
                  action="/admin/papel?faces-redirect=true"
                  rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL')}" />
                <p:menuitem value="#{msg['professor.listagem.descricao']}"
                  action="/admin/professor?faces-redirect=true"
      rendered="#{sec:areAnyGranted('ROLE_ADMIN,ROLE_RESPONSAVEL')}" />
                <p:menuitem
value="#{msg['profissional.listagem.descricao']}"
                  action="/admin/profissional?faces-redirect=true"
      rendered="#{sec:areAnyGranted('ROLE_ADMIN,ROLE_RESPONSAVEL')}" />
                <p:menuitem value="#{msg['projeto.listagem.descricao']}"
                  action="/admin/projeto?faces-redirect=true"
      rendered="#{sec:areAnyGranted('ROLE_RESPONSAVEL,ROLE_ALUNO,ROLE_AVALI
ADOR')}" />
                <p:menuitem
value="#{msg['tipoprojeto.listagem.descricao']}"
                  action="/admin/tipoprojeto?faces-redirect=true"
                        rendered="#{sec:areAnyGranted('ROLE_ADMIN')}" />
                <p:menuitem value="#{msg['usuario.listagem.descricao']}"
                  action="/admin/usuario?faces-redirect=true"
                        rendered="#{sec:areAnyGranted('ROLE_ADMIN')}" />
            </p:submenu>
        </p:panelMenu>
    </h:form>
</ui:fragment>
```

Listagem 16 – menu.xhtml

O JSF oferece também a possibilidade da criação de componentes customizados, também chamados de “Composite Components”, a Listagem 17 apresenta o código do arquivo “tabelaCrudBase.xhtml”.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:cc="http://java.sun.com/jsf/composite">

<!-- INTERFACE -->
<cc:interface>
      <cc:attribute name="controller"
type="br.com.joaopichetti.controller.CrudBaseController"
            required="true" />
</cc:interface>

<!-- IMPLEMENTATION -->
<cc:implementation>
      <p:dataTable id="tblDados" var="data"
            value="#{cc.attrs.controller.lista}" resizableColumns="true"

      emptyMessage="#{msg[cc.attrs.controller.domainName.concat('.listagem.
tabela.vazia')]}" rows="10"
            rowsPerPageTemplate="10,15,20" paginatorPosition="bottom"
            paginator="true"
            paginatorTemplate="{FirstPageLink} {PreviousPageLink}
{PageLinks}
                              {NextPageLink} {LastPageLink}
{RowsPerPageDropdown}">
            <p:columns value="#{cc.attrs.controller.colunas}"
                  var="coluna" headerText="#{msg[coluna.codDesc]}"
sortBy="#{data[coluna.campoSort]}"
                  width="#{coluna.largura}">
                        <h:outputText value="#{data[coluna.campo]}" />
            </p:columns>
            <!-- Colunas adicionais -->
            <cc:insertChildren></cc:insertChildren>
      </p:dataTable>
</cc:implementation>
</html>
```

Listagem 17 – tabelaCrudBase.xhtml

Como visto na Listagem 17, o arquivo “tabelaCrudBase” é um componente customizado, sua criação envolve duas *tags* principais, a *tag* “interface”, na qual declara-se os atributos e a *tag* “implementation”, na qual é implementada a criação do componente. Esse componente utiliza o componente “dataTable” do PrimeFaces e dentro do “dataTable” utiliza-se o componente “columns”, também do PrimeFaces. O componente “columns” é utilizado para criar as colunas da tabela dinamicamente, bastando passar uma lista de objetos (que fica no controlador) como parâmetro do atributo “value”. A criação das colunas no controlador foi citada anteriormente na Listagem 8. A Listagem 18 apresenta o código de outro componente customizado, o componente “telaCrudBase”.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
```

```
      xmlns:cc="http://java.sun.com/jsf/composite"
      xmlns:j="http://xmlns.jcp.org/jsf/composite/components">
<!-- INTERFACE -->
<cc:interface>
      <cc:attribute name="controller"
type="br.com.joaopichetti.controller.CrudBaseController"
            required="true" />
</cc:interface>

<!-- IMPLEMENTATION -->
<cc:implementation>
      <p:panel
header="#{msg[cc.attrs.controller.domainName.concat('.listagem.descricao')]
}" styleClass="sem-borda">
            <j:toolbarCrudBase controller="#{cc.attrs.controller}" />
            <j:tabelaCrudBase controller="#{cc.attrs.controller}">
                  <p:column headerText="#{msg['comum.acoes']}" width="75">

                        <j:edicaoCrudBase
controller="#{cc.attrs.controller}" />
                        <j:exclusaoCrudBase
controller="#{cc.attrs.controller}" />
              </p:column>
            </j:tabelaCrudBase>
            <j:dialogCrudBase controller="#{cc.attrs.controller}">
                  <cc:insertChildren>
                        <!-- Aqui ficara o conteudo do formulario definido
em cada pagina -->
                  </cc:insertChildren>
            </j:dialogCrudBase>
      </p:panel>
</cc:implementation>
</html>
```

Listagem 18 – telaCrudBase.xhtml

Como visto na Listagem 18, esse componente é responsável por criar uma tela para a manutenção básica (operações CRUD) das entidades. Esse componente faz uso do componente “tabelaCrudBase” (Listagem 17) e de outros componentes customizados. Os componentes customizados encontram-se no diretório “resources/components”.

Utilizando o componente “telaCrudBase”, a implementação das telas de manutenção diminui consideravelmente, como pode ser visto na Listagem 19, que mostra o conteúdo do arquivo “curso.xhtml”.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:p="http://primefaces.org/ui"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:j="http://xmlns.jcp.org/jsf/composite/components"
                template="/resources/templates/base.xhtml">
```

```
    <ui:define name="conteudo">
      <j:telaCrudBase controller="#{cursoController}">
            <h:panelGrid id="pnlForm" columns="2" cellspacing="5">
                  <p:outputLabel value="#{msg['comum.nome']}" for="nome" />
                        <p:inputText id="nome"
value="#{cursoController.instancia.nome}"
                              maxlength="60" size="70" required="true"

      requiredMessage="#{msg['comum.nome.required']}" />

                        <p:outputLabel value="#{msg['comum.descricao']}"
for="descricao" />
                        <p:inputText id="descricao"
value="#{cursoController.instancia.descricao}"
                              maxlength="100" size="110" />
              </h:panelGrid>
            </j:telaCrudBase>
    </ui:define>
</ui:composition>
```

Listagem 19 – curso.xhtml

Como visto na Listagem 19, para implementar uma tela de manutenção, basta utilizar o componente “telaCrudBase”, passando por parâmetro o controlador referente à entidade que será gerenciada. Todas as telas protegidas da aplicação ficam no diretório “admin”.

Mesmo com a aplicação utilizando o Spring Boot, ainda é necessário realizar algumas configurações realizadas por meio de arquivos XML, isso é necessário pois a aplicação também utiliza o JSF. A Listagem 20 mostra o código do arquivo “springsecurity.taglib.xml”.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE facelet-taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
"http://java.sun.com/dtd/facelet-taglib_1_0.dtd">
<facelet-taglib>
      <namespace>http://www.springframework.org/security/tags</namespace>
      <tag>
            <tag-name>authorize</tag-name>
            <handler-
class>org.springframework.faces.security.FaceletsAuthorizeTagHandler</handl
er-class>
      </tag>
      <function>
            <function-name>areAllGranted</function-name>
            <function-
class>org.springframework.faces.security.FaceletsAuthorizeTagUtils</functio
n-class>
            <function-signature>boolean
areAllGranted(java.lang.String)</function-signature>
      </function>
      <function>
            <function-name>areAnyGranted</function-name>
            <function-
class>org.springframework.faces.security.FaceletsAuthorizeTagUtils</functio
n-class>
            <function-signature>boolean
areAnyGranted(java.lang.String)</function-signature>
```

```
      </function>
      <function>
            <function-name>areNotGranted</function-name>
            <function-
class>org.springframework.faces.security.FaceletsAuthorizeTagUtils</functio
n-class>
            <function-signature>boolean
areNotGranted(java.lang.String)</function-signature>
      </function>
      <function>
            <function-name>isAllowed</function-name>
            <function-
class>org.springframework.faces.security.FaceletsAuthorizeTagUtils</functio
n-class>
            <function-signature>boolean
isAllowed(java.lang.String,java.lang.String)</function-signature>
      </function>
</facelet-taglib>
```

Listagem 20 – springsecurity.taglib.xml

O arquivo "springsecurity.taglib.xml" (Listagem 20) serve para a utilização das *tags* de segurança do Spring, não disponíveis nativamente em páginas JSF. Com essas *tags* é possível renderizar, ou não, partes da página de acordo com o perfil do usuário logado. Um dos arquivos que utiliza as *tags* de segurança é o arquivo "menu.xhtml" (Listagem 16). Outro arquivo de configuração pode ser visto na Listagem 21, o arquivo "faces-config.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">

    <application>
      <resource-bundle>
            <base-name>messages</base-name>
            <var>msg</var>
      </resource-bundle>

      <el-
resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-
resolver>

      <system-event-listener>
                <system-event-listener-
class>br.com.joaopichetti.scope.ViewScopeCallbackRegistrer</system-event-
listener-class>
                <system-event-
class>javax.faces.event.PostConstructViewMapEvent</system-event-class>
                <source-class>javax.faces.component.UIViewRoot</source-
class>
            </system-event-listener>

            <system-event-listener>
```

```
                <system-event-listener-
class>br.com.joaopichetti.scope.ViewScopeCallbackRegistrer</system-event-
listener-class>
                <system-event-
class>javax.faces.event.PreDestroyViewMapEvent</system-event-class>
                <source-class>javax.faces.component.UIViewRoot</source-
class>
            </system-event-listener>
    </application>

</faces-config>
```

Listagem 21 – faces-config.xml

Por meio da *tag* "resource-bundle" (Listagem 21), é possível registrar uma variável, que é utilizada nas telas para a realização da internacionalização. Já a *tag* "el-resolver" é utilizada para registrar uma classe do Spring que fará a resolução da "Expression Language", utilizada pelo JSF. E a *tag* "system-event-listener" é utilizada para registrar as classes referentes à implementação do escopo customizado. Esses arquivos de configuração encontram-se no diretório "WEB-INF".

5 CONCLUSÃO

O objetivo deste trabalho foi implementar um aplicativo *web* para registrar a avaliação de trabalhos acadêmicos apresentados perante bancas. O aplicativo foi desenvolvido utilizando-se diversas ferramentas e tecnologias. Foi necessário implementar diversas regras e restrições relacionadas à disponibilização da versão para a banca, tornando o processo de implementação mais demorado do que o inicialmente planejado.

Uma dessas tecnologias utilizada no desenvolvimento do sistema é o Spring Boot. Essa ferramenta facilita o desenvolvimento de aplicações Java, pois sua configuração é feita por meio de classes Java e anotações e por ser um dos *frameworks* mais utilizados, encontra-se muita documentação e exemplos de utilização. Por meio do módulo Spring Data JPA, a persistência de dados se torna extremamente simples. E por meio do módulo Spring Security a aplicação fica segura possuindo autenticação e autorização. O Spring é facilmente integrado com o JSF, mas deixa a desejar em relação aos escopos dos controladores, pois não há o escopo de visão, muito utilizado em aplicações JSF. Dessa forma deve-se implementar o escopo de visão para ser utilizado pelo Spring, o que é um pouco complexo.

Outra tecnologia utilizada e considerada relevante é o PrimeFaces, biblioteca que implementa a especificação do JSF. O PrimeFaces, assim como outras bibliotecas semelhantes, tem a finalidade de melhorar os componentes já existentes na especificação e, ao mesmo tempo, incluir novos. Ele possui uma aceitação muito boa pela comunidade e uma vasta documentação, que pode ser encontrada, inclusive, na página oficial (http://primefaces.org/), facilitando o desenvolvimento e a solução de possíveis problemas. Uma das suas desvantagens é a estilização visual da aplicação, pois, criar um estilo diferente do padrão é bastante complexo, limitando-se, o programador, geralmente, a customizar o já existente.

Como trabalhos futuros, ressaltam-se a implementação de algumas funcionalidades, como o envio automático de e-mail para os participantes da banca, o aviso de avaliações pendentes, bem como a geração automática da ata de defesa após o projeto ser avaliado.

REFERÊNCIAS

COMAI, Sara; CARUGHI, Giovanni Toffetti. A behavioral model for rich internet applications. **Web Engineering Lecture Notes in Computer Science**, v. 4607, p. 364-369, 2007.

DWORAK, Hugo. **A concept of a web application blending thin and fat client architectures**. Fourth International Conference on Dependability of Computer Systems, 2009, p. 84-90.

ICEFACES. **IceFaces**. Disponível em: <http://icefaces.org/>. Acesso em: 28 set. 2015.

MELIÁ, Santiago; GÓMEZ, Jaime; PÉREZ, Sandy; DÍAZ, Oscar. Architectural and technological variability in rich internet applications. **IEEE Internet Computing**, may/june 2010, 2010, p. 24-32.

MOJARRA. **Oracle Mojarra**. Disponível em: <http://javaserverfaces.java.net/>. Acesso em: 28 set. 2015.

MYFACES. **Apache MyFaces**. Disponível em: <http://myfaces.apache.org/>. Acesso em: 28 set. 2015.

PAVLIĆ, Daniel; PAVLIĆ, Mile; JOVANOVIĆ, Vladan. **Future of internet technologies.** MIPRO 2012, p. 1366-1371.

PRIMEFACES. **PrimeFaces**. Disponível em: <http://primefaces.org/>. Acesso em: 28 set. 2015.

RICHFACES. **RichFaces**. Disponível em: <http://richfaces.org/>. Acesso em: 28 set. 2015.