UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JOÃO GUILHERME BRASIL PICHETTI

SISTEMA WEB PARA GERENCIAMENTO DE BANCAS DE TRABALHOS ACADÊMICOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

JOÃO GUILHERME BRASIL PICHETTI

SISTEMA WEB PARA GERENCIAMENTO DE BANCAS DE TRABALHOS ACADÊMICOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2013

ATA Nº: **225**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **JOÃO GUILHERME BRASIL PICHETTI**.

Às 13:58 hrs do dia 20 de fevereiro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Eliane Maria de Bortoli Fávero (Convidada) e Rúbia E. O. Schultz Ascari (Convidada), para avaliar o Trabalho de Diplomação do aluno João Guilherme Brasil Pichetti, matrícula 1261479, sob o título **Sistema Web para Gerenciamento de Bancas de Trabalhos Acadêmicos**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 14:18 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Eliane Maria de Bortoli Fávero, M.Sc.
Convidada

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Eliane Maria de Bortoli Fávero, M.Sc
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

PICHETTI, João Guilherme Brasil. Sistema web para gerenciamento de bancas de trabalhos acadêmicos. 2013. 67 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

As atividades acadêmicas como apresentações de estágio curricular obrigatório e de trabalho de conclusão de curso ocorrem perante uma banca. A banca, composta por professores da própria instituição, de outras instituições e mesmo de pessoas da comunidade, avalia o trabalho. A avaliação é composta por itens com nota e peso individuais. Esses itens podem ser agrupados compondo áreas de avaliação. Dessa avaliação é obtida uma nota (média) que define se o aluno está aprovado ou não na referida atividade. Uma ata é elaborada para registrar o resultado da banca para o trabalho. Atualmente, no Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, a composição das bancas, a avaliação dos trabalhos e a geração da ata e do relatório de participantes, são atividades realizadas em sua maioria manualmente. Como resultado deste trabalho foi realizada a implementação de um aplicativo que permitisse automatizar várias dessas tarefas. Como forma de facilitar o acesso, o aplicativo foi desenvolvido com tecnologias para *web*.

Palavras-chave: Aplicativo web. Java para web. Bancas de estágio e TCC.

ABSTRACT

PICHETTI, João Guilherme Brasil. Web system to manager evaluatons of internship report and completion of course work. 2013. 67 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

Academic activities, such as internship and completion course work, are presented to examining board. Teachers and professionals evaluate the presentation, the text and the work. The form of evaluation is composed by items with evaluation score and grade. Each evaluation score has a grade. These items can be grouped in to areas of evaluation. The result of all evaluations defines an average for each student. Now, in the Academic Department of Informatics of the Federal Technological University of Paraná, Campus Pato Branco, the examining board, the evaluation and generation of reports are made manually. Thus, a web system was developed to help these evaluations. Aiming to facilitate the access, the software was developed with web technologies.

Keywords: Web application. Web Java. Management of evaluation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

AIR	*Adobe Integrated Runtime*
CSS	*Cascading Style Sheets*
DAO	*Data Access Object*
HTML	*HiperText Markup Language*
JSF	*Java Server Faces*
ORM	*Object Relational Mapping*
RIA	*Rich Internet Application*
SQL	*Structured Query 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.

1.1 CONSIDERAÇÕES INICIAIS

Atividades acadêmicas como, por exemplo, estágios supervisionados ou obrigatórios, trabalhos de conclusão de curso e propostas de trabalho de conclusão de curso são avaliadas por apresentação perante banca. Uma banca é composta por professores da própria Universidade ou de outras Instituições de Ensino e pode ser composta, também, por 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.

Na Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, atualmente a avaliação desses trabalhos é realizada por meio de fichas impressas 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. As notas que são atribuídas por cada um dos membros da banca nas fichas precisam ser digitadas pelo professor responsável pela atividade, para assim obter a média e elaborar 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. Esse aplicativo foi desenvolvido como resultado da realização deste trabalho.

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

Desenvolver um aplicativo computacional para gerenciar avaliações de trabalhos acadêmicos apresentados perante bancas.

1.2.2 Objetivos Específicos

- Facilitar o registro de avaliação de trabalhos acadêmicos que são apresentados perante bancas avaliadoras.
- Agilizar o processo de registro da avaliação de bancas de trabalhos acadêmicos.
- Facilitar o processo de emissão de comprovante aos professores de participação em bancas de trabalhos acadêmicos.
- Facilitar o processo de emissão de atas de defesa de trabalhos acadêmicos apresentados para 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* tornaria mais fácil o processo de avaliação. Para os professores responsáveis pelas atividades, além de não ser necessário realizar cálculos e lançamentos, seria 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 haveria 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 conceitos que fundamentam a proposta do trabalho que está centrado na modelagem de um sistema *web* e na apresentação das tecnologias utilizadas para implementar o mesmo. Como o sistema será para *web* e implementado utilizando tecnologias que caracterizam a interface do sistema como rica, o referencial teórico está fundamentado em aplicações Internet rica, as denominadas *Rich Internet Application* (RIA).

2.1 CONTEXTO CONCEITUAL

A arquitetura distribuída da *web* tem herdado os benefícios das aplicações em rede como baixos custos de manutenção, descentralização e compartilhamento de recursos (LINAJE et al., 2007). Porém, a complexidade das atividades realizadas por meio das interfaces *web* mantém crescente a necessidade de interação semelhante às aplicações *desktop*. Isso porque as aplicações *desktop* apresentam recursos de interface considerados muitos superiores (em termos de funcionalidades que facilitam o trabalho do usuário) aos das aplicações *web* baseadas em *HiperText Markup Language* (HTML) com páginas vinculadas por meio de *links*.

Em uma aplicação dinâmica baseada em HTML, denominada como tradicional, a interface da aplicação é um documento HTML processado pelo servidor a cada requisição do usuário (COMAI; CARUGHI, 2007). Quando o usuário interage com a página, seguindo *hiperlinks* ou enviando dados de formulários, o servidor é invocado e a página de destino é processada a partir de uma página padrão que é retornada para o cliente. O papel do cliente é interceptar as ações do usuário, enviar as requisições para o servidor e apresentar as respostas.

Neste contexto, aplicações *web* baseadas em HTML estão apresentando os seus limites em relação aos altos níveis de interação e suporte a multimídia necessários às aplicações. Uma solução para esse problema, indicada por Linaje et al. (2007) reside na combinação dos benefícios do modelo de distribuição *web* com a interatividade da interface das aplicações *desktop* definindo as aplicações Internet ricas, denominadas *Rich Internet Applications*.

Nas aplicações RIA, para o cliente é atribuída uma fração dos dados e da computação. Assim o usuário pode realizar interações complexas com a interface sem invocar o servidor e uma quantidade menor de dados precisa ser trocada. Além disso, se a interação do usuário

requer uma requisição ao servidor e uma resposta deste para o cliente para atualizar alguns dados, o cliente pode seletivamente receber do servidor somente a informação que precisa ser trocada, atualizando seu estado interno e reapresentando o conteúdo modificado (COMAI; CARUGHI, 2007).

Os recursos de interatividade das RIAs fazem com que os aplicativos baseados na Internet sejam mais fáceis de usar e mais funcionais e também que provejam uma solução para problemas com aplicações *web* tradicionais como baixo desempenho e interatividade limitada (LAWTON, 2008). Meliá et al. (2010) complementam que as RIAs estão oferecendo aplicações mais responsivas ocasionando uma experiência melhor para o usuário do que a oferecida por aplicações *web* tradicionais. Para Rossi et al. (2008), as RIAs combinam a simplicidade do hipertexto com a flexibilidade da interface *desktop*.

2.2 MODELO DE APLICAÇÃO WEB TRADICIONAL

Uma aplicação *web* tradicional dinâmica é descrita por meio de sua estrutura e comportamento (COMAI; CARUGHI, 2007). O comportamento define o modelo de dados que especifica o conteúdo dos objetos da aplicação e a estrutura define o modelo de interface (também denominado como modelo de hipertexto) que descreve o *front-end* da aplicação para a interação com o usuário.

O modelo de interface define o conteúdo das páginas e o mecanismo que suporta a navegação e a interação com o usuário. Os seus elementos chave são (COMAI; CARUGHI, 2007):

a) Componentes de conteúdo – são usados para extrair o conteúdo armazenado na camada de dados.

b) Contêineres de interface – são coleções estruturadas de componentes representando páginas *web* ou sub-módulos de páginas *web* que podem ser organizadas hierarquicamente.

c) Parâmetros – representados pelas regras de passagem de parâmetros e expressa a dependência entre um par de componentes: consiste de um componente de origem e um componente de destino e um mapeamento entre os parâmetros de saída da origem e os parâmetros de entrada do destino.

d) Mecanismos de interação – são especificados por meio de *links*, que em aplicações *web* tradicionais representam *hiperlinks* e comandos de entrada, representados tipicamente por âncoras ou botões em formulários.

O modelo dinâmico explicita o que ocorre quando o usuário (COMAI; CARUGHI, 2007) acessa uma página ou navega por *links* internos e ativa uma sequência de componentes de negócio.

2.3 MODELO DE APLICAÇÃO WEB COM INTERFACE RICA

As RIAs são aplicações cliente/servidor que surgiram da intersecção de duas culturas de desenvolvimento concorrentes: aplicações *web* e *desktop* (MELIA et al., 2010). As RIAs se beneficiam das características de distribuição do processamento entre cliente e servidor e manutenção da *web*, ao mesmo tempo que suportam uma interface com o usuários mais rica, como as aplicações *desktop*.

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 mostrado, mas tornou-se inconsistente com o restante da página.

O Quadro 1 apresenta as características consideradas no projeto de um cliente RIA de acordo com o indicado no trabalho de Preciado et al. (2007) e Brambilla (2008).

Característica	**Descrição**
Trabalho *offline*	Habilidade de trabalhar enquanto desconectada para *download* e pode armazenar dados no lado cliente. Esta característica requer processamento da lógica de negócio e mecanismo de armazenamento no lado cliente.
Armazenamento	Facilidade de armazenamento do lado cliente que manipula os dados que vem do servidor. Esse armazenamento pode ser persistente ou volátil.
Cache	Habilidade de manter informações do servidor durante um determinado período de tempo, melhorando o desempenho da aplicação e a capacidade de resposta da interface com o usuário.
Lógica de negócio	Capacidade de processamento melhorada, permitindo à camada cliente desempenhar processos complexos. Essa característica possui um atributo de localização que determina se é um cliente,

	um servidor de lógica de negócio ou ambos.
Manipulador de eventos	Uma coreografia baseada em eventos entre componentes de interface com o usuário que podem ser sincronizados com um barramento de eventos centralizado ou gerenciador de padrões descentralizado.
Validação	Regras de validação para entradas do usuário bem como para as regras de negócio do cliente.
Templates	Possibilidade de suporte para criar visões e o leiaute da apresentação em tempo de execução.
Plataforma	Possibilidade de determinar qual plataforma é usada para implementar a camada do cliente.

Quadro 1 – Características do cliente em RIAs
Fonte: traduzido de Meliá et al. (2010, p. 27).

Lawton (2008) ressalta que por volta de 2005 os desenvolvedores estavam aprendendo a usar Ajax para criar interface de aplicações *web* semelhante à interface de aplicações *desktop*, tal como o Google Maps. A partir disso vários vendedores têm lançado suas próprias plataformas para desenvolvimento de RIAs baseadas em Ajax, como ASP.Net da Microsoft e Google Web Toolkit. E outras empresas lançaram plataformas não baseadas em Ajax, como Flash da Adobe e *Adobe Integrated Runtime* (AIR), Silverlight da Microsoft e JavaFx da Sun.

Essa diversidade de tecnologias define o desafio indicado por Meliá et al. (2010) que é o de selecionar as alternativas corretas dentre as existentes para implementar RIAs de forma a satisfazer os requisitos do usuário e os definidos para a aplicação, incluindo os relacionados ao desempenho.

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 ferramentas utilizadas e o método apresenta a sequência das principais atividades realizadas.

3.1 MATERIAIS

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

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Astah* Community	6.2.1 (model version: 33)	http://astah.net/editions/community	Documentação da modelagem baseada na UML.
Case Studio 2	2.25	http://www.casestudio.com	Modelagem do diagrama de entidades e relacionamentos do banco de dados.
Linguagem Java	JDK 1.7	http://www.oracle.com	Linguagem de programação.
Eclipse Juno	4.2.1	http://www.eclipse.org/	Ambiente de desenvolvimento.
MySQL	5	http://www.mysql.com/	Banco de dados.
MySQL WorkBench	5.2 CE	http://www.mysql.com/products/workbench/	Administrador do banco de dados.
Apache Tomcat	7.0	http://www.apache.org/	Servidor *web* para a aplicação.
PrimeFaces	4.0	http://primefaces.org/	Biblioteca de componentes para *web*.
JSF	2.0	https://javaserverfaces-spec-public.java.net/	*Framework* para *web.*
Hibernate	3	http://www.hibernate.org/	Efetuar o mapeamento objeto-relacional e a persistência dos dados.

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

A modelagem e a implementação do aplicativo *web* para avaliação de atividades acadêmicas realizadas por meio de banca de avaliadores têm como base o modelo sequencial linear descrito em Pressman (2008), complementado pelo processo unificado (BLAHA et al., 2006). O modelo sequencial foi utilizado para denominar os processos (ou fases). O processo unificado auxiliou na definição dos ciclos iterativos de modelagem e de implementação.

Ressalta-se que os processos de requisitos e de análise e projeto foram realizados como trabalho de estágio supervisionado pelo autor deste trabalho de conclusão de curso. Como trabalho de estágio também ocorreu o estudo das tecnologias que foi representado pela implementação das operações de inclusão, exclusão, consulta e alteração de um cadastro.

O Quadro 2 apresenta os processos (fluxos de trabalho) e as iterações desenvolvidas. Essas iterações complementam as atividades realizadas como trabalho de estágio.

Iterações / Processos	1ª iteração	2ª iteração	3ª iteração
Requisitos	Revisão dos requisitos.		
Análise e projeto	Revisão da modelagem.	Ajustes no diagrama de entidades e relacionamentos	
Implementação	Dos cadastros.	Da composição de uma apresentação, lançamento de notas, composição de modelos de avaliação.	Da composição da banca.
Testes	De código, realizados pelo programador (o autor do trabalho).	De código, realizados pelo programador (o autor do trabalho). Das funcionalidades do sistema realizadas pelos responsáveis pelas atividades de estágio e trabalho de conclusão de curso.	De código, realizados pelo programador (o autor do trabalho). De funcionalidades e interação realizados pela orientadora.

Quadro 2 – Iterações definidas

A seguir estão descritas as etapas (identificadas como processos no Quadro 2) definidas para o desenvolvimento do aplicativo e as principais atividades de cada uma dessas etapas.

a) Levantamento de requisitos

O levantamento dos requisitos foi realizado como trabalho de estágio pelo autor deste trabalho de conclusão de curso. Como resultado da avaliação do trabalho de estágio os

requisitos foram revistos. Aspectos relacionados à apresentação e composição da banca foram repensados gerando ajustes no banco de dados.

b) Análise e projeto do sistema

Como resultado da avaliação do trabalho de estágio a modelagem foi revisada. Ajustes foram realizados de forma a atender as observações da professora responsável pela atividade de trabalho de conclusão de curso.

c) Implementação

A implementação foi realizada utilizando a ferramenta Eclipse Juno. Como trabalho de estágio foram implementadas as funcionalidades de cadastros simples visando exemplificar o uso das tecnologias e o aprendizado das mesmas. Em termos de interface, o objetivo foi experimentar e testar a melhor forma de compor os formulários e de disponibilizar as informações na tela. Como trabalho de conclusão de curso o restante dos requisitos foi implementado.

d) Testes

Os testes foram informais e realizados à medida que as iterações ocorriam. Esses testes incluíram verificação do código, da forma de interação com o aplicativo e de suas funcionalidades. Os testes de código foram realizados pelo autor deste trabalho de maneira informal, sem um plano de testes definido. As funcionalidades e a interação com o sistema foram testadas pela orientadora e pela professora responsável pela coordenação de trabalho de conclusão de curso.

4 RESULTADO

Este capítulo apresenta o resultado deste trabalho que é a implementação de um aplicativo para registrar a avaliação de trabalhos acadêmicos apresentados perante bancas. No capítulo constam códigos que visam exemplificar como a implementação foi realizada. A modelagem foi realizada como trabalho de estágio pelo autor deste trabalho, mas consta neste capítulo como forma de facilitar o entendimento dos requisitos do aplicativo implementado.

4.1 ESCOPO DO SISTEMA

O aplicativo computacional implementado como resultado deste trabalho automatiza atividades do procedimento de avaliação de trabalhos acadêmicos que são apresentados perante bancas. 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 pode ser realizado por mais de um aluno. Contudo cada aluno é avaliado individualmente por cada um dos membros que compõem a banca. A avaliação é realizada por 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. É comum que a quantidade de componentes de uma banca seja definida de acordo com o tipo de trabalho.

O professor responsável pela atividade de estágio, trabalho de conclusão de curso ou outra atividade que é avaliada pelo sistema de banca de avaliadores, faz a composição da banca para o respectivo trabalho. Assim, o avaliador terá acesso para avaliar somente os trabalhos dos quais ele é membro de banca. O professor responsável pelo trabalho de conclusão de curso somente terá acesso às bancas de trabalhos e propostas de conclusão de curso. O professor responsável pela atividade de estágio somente terá acesso aos trabalhos de estágio. Ressalta-se que o tipo de projeto (estágio, trabalho de conclusão de curso, proposta) é cadastrado, ou seja, não está predefinido no aplicativo.

4.2 MODELAGEM DO SISTEMA

O diagrama conceitual de domínio é apresentado na Figura 1. 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.

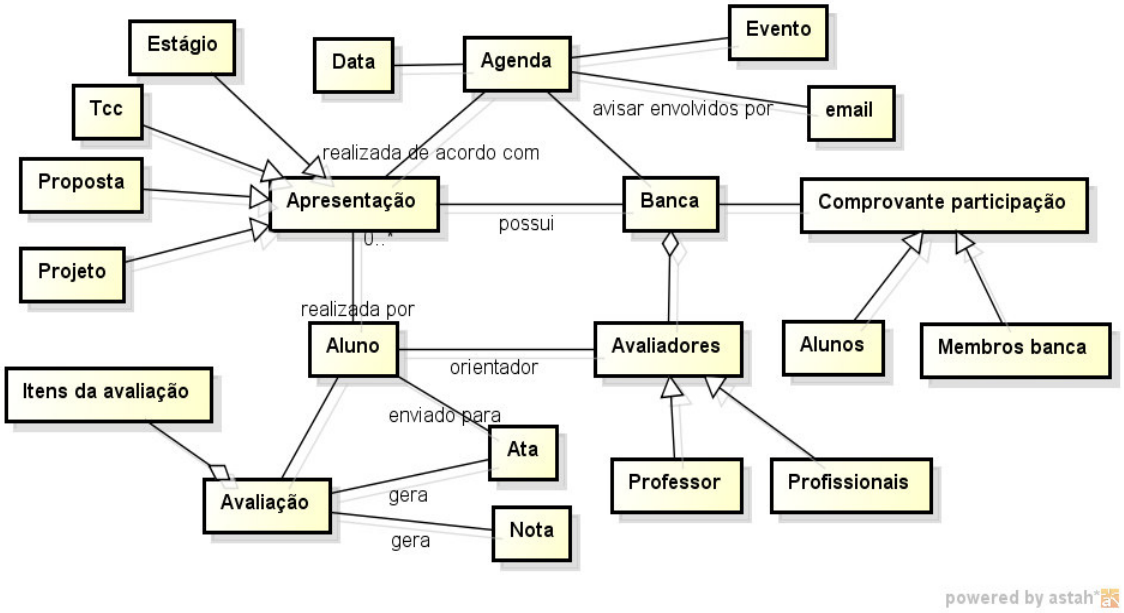

Figura 1 – Modelo conceitual de domínio do sistema

De acordo com a representação da Figura 1, uma apresentação pode ser de vários tipos como estágio e TCC, esse apresentação será realizada por um aluno e ocorre perante uma banca. A avaliação do aluno é composta por itens que geram uma nota e uma ata de defesa/apresentação. Uma banca é composta por avaliadores. Um dos avaliadores é o orientador do aluno, os demais avaliadores são professores ou profissionais. Os participantes da banca recebem comprovante de participação. Alunos que assistem a 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 que é vinculada a uma da (dia e horário). A apresentação gera um evento e os envolvidos são avisados da mesma por meio de *email*.

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

Identificação	Nome	Descrição
RF01	Cadastrar apresentação	Uma banca é composta para cada apresentação. Apresentação se refere ao evento de apresentação do trabalho perante uma banca de avaliação.
RF02	Cadastrar projeto	Cadastrar os projetos que são apresentados perante banca. Um projeto é de um tipo: estágio, tcc, projeto, proposta, dentre outros.
RF03	Cadastrar alunos	Os alunos que irão apresentar o trabalho.
RF04	Cadastrar ouvintes	As pessoas que assistem à apresentação recebem comprovante de participação. O comprovante pode ser na forma de um a listagem com todos os ouvintes.
RF05	Cadastrar componentes da banca	Os componentes (professores e convidados) das bancas de avaliação. São os avaliadores dos trabalhos.
RF06	Cadastrar grupos de itens de avaliação	Os grupos de itens que comporão a avaliação.
RF07	Cadastrar itens de avaliação	Os itens que serão avaliados. Os itens serão organizados por grupos.
RF08	Compor avaliação	Selecionar os grupos e seus respectivos itens que comporão a avaliação. Um modelo de avaliação é utilizado pelos membros da banca para realizar a avaliação. É a ficha de avaliação.
RF09	Compor banca	Uma banca é composta por avaliadores (professores e outros) e pelos alunos que apresentam o trabalho. Uma banca possui data e horário previstos para realização. E data, horário de início e fim de realização, quando a apresentação foi efetivamente realizada. Uma banca possui ouvintes que podem receber comprovante de participação.
RF10	Realizar avaliação	Cada membro da banca realiza a avaliação de cada um dos componentes da equipe que está apresentando o trabalho. A avaliação é composta por itens e cada item possui uma nota com peso associado que definem a média (nota final) do aluno. Após todos os professores realizarem a avaliação é gerado um relatório que é enviado por *email*, para o(s) aluno(s) e o orientador(a). Os membros da banca também recebem um comprovante de todas as bancas que participaram (como orientadores ou como avaliadores).
RF11	Emitir ata	Emitir ata comprobatória de realização da banca. A ata deve atender o formato utilizado pelo curso. A ata, após assinada pelos membros da banca, é digitalizada e é anexada ao relatório ou monografia, além de ser enviada para o aluno que apresentou o trabalho.
RF12	Emitir relatório de apresentações	Emitir um relatório das apresentações por período de data. Esse relatório será utilizado para divulgar as bancas. O relatório será composto por aluno: título do

		trabalho, data, horário e local de apresentação, orientador e professores componentes da banca.
RF13	Emitir comprovante de orientação	Emitir comprovante dos professores orientadores por período de data.
RF14	Emitir comprovante de participação em banca	Emitir comprovante para os avaliadores por período de data. O comprovante deve constar o título do trabalho, os autores do mesmo, a data de apresentação e a função do professor.
RF015	Cadastrar cursos	Cursos aos quais pertencem os alunos que apresentam os trabalhos perante bancas de avaliação.
RF016	Cadastrar usuários	Usuários com acesso ao sistema. Os usuários possuem permissões distintas. Administrador com acesso a todas as funcionalidades do sistema. Professor responsável com acesso a todas as bancas dos projetos aos quais ele tem acesso (tipo de projeto). Avaliadores que são os membros das bancas dos trabalhos apresentados.
RF17	Compor modelo de avaliação	Um modelo de avaliação é composto a partir de um conjunto de itens de avaliação. Esses itens podem ter pesos associados, definindo uma média ponderada.
RF18	Cadastrar papéis	São os papéis que os avaliadores realizam em um trabalho: orientador, supervisor, convidado, dentre outros.

Quadro 3 – Requisitos funcionais

A listagem do Quadro 4 apresenta os requisitos não-funcionais identificados para o sistema como um todo, também denominados de requisitos suplementares. Os requisitos não funcionais explicitam regras de negócio, restrições ao sistema de acesso, requisitos de qualidade, desempenho e segurança, dentre outros.

Identificação	Nome	Descrição
RNF01	Acesso ao sistema	O acesso ao sistema será realizado por meio de *login* e senha.
RNF02	Avaliação	Cada membro da banca poderá ter acesso para avaliar somente os alunos das bancas das quais ele faz parte.
RNF02	Acesso avaliação	A avaliação será liberada somente na data e horário definidos para a realização da banca e permanece aberta para o respectivo avaliador até que a avaliação seja realizada.
RNF03	Banca (apresentação)	Uma apresentação é de um tipo, realizada por um ou mais alunos, avaliada por uma banca e realizada de acordo com uma agenda. Mesmo que uma apresentação seja realizada por mais de um aluno, cada aluno é avaliado individualmente. Os componentes da banca e o(s) aluno(s) que apresenta(m) o trabalho são avisados por *email* da data da apresentação. Os membros da banca recebem comprovante de

		participação. Assim, como os alunos que assistem à apresentação.

Quadro 4 – Requisitos não funcionais

O diagrama de casos de uso apresentado na Figura 2 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: avaliadores, professor responsável e administrador. O Administrador é responsável pelos cadastros de usuários do sistema. O professor responsável é quem realiza a atividade de compor as bancas, agendar as defesas, emitir as atas e registrar o resultado das bancas no sistema acadêmico. Os avaliadores são professores ou profissionais que compõem as bancas avaliando os trabalhos apresentados. O professor responsável somente tem acesso ao tipo de trabalho aos quais ele é o responsável.

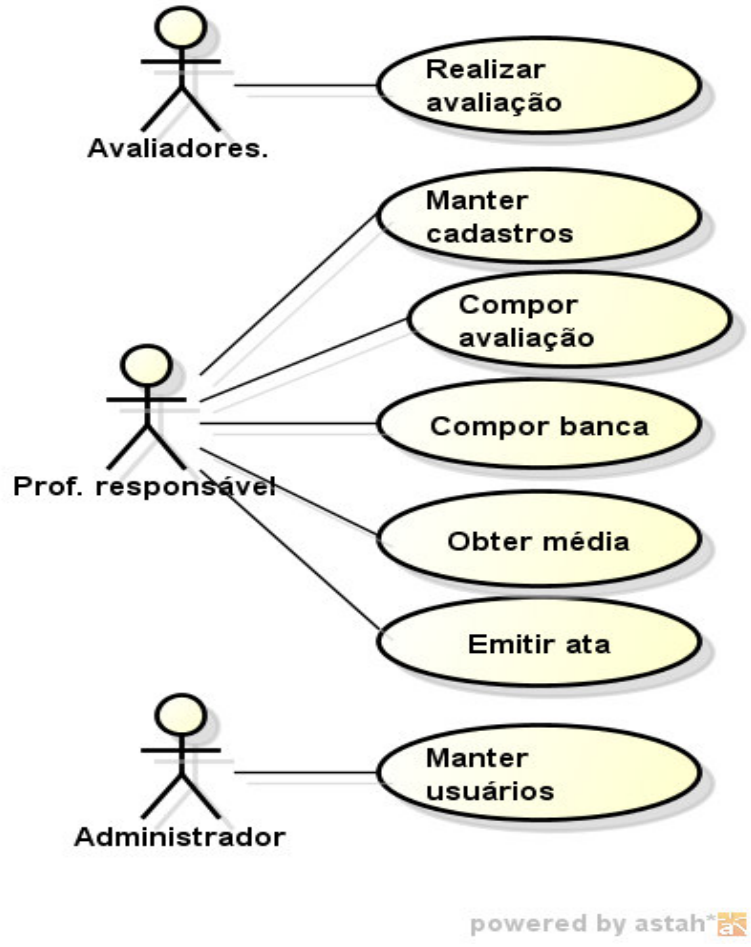

Figura 2 – Diagrama de casos de uso

No Quadro 5 está a descrição do caso de uso realizar avaliação.

Caso de uso: Realizar avaliação **Descrição**: Durante o evento de apresentação do trabalho ou posteriormente o avaliador realiza o

lançamento do resultado da sua avaliação no sistema.

Evento Iniciador:

Data igual ou posterior à data armazenada para realização da banca.

Atores:

Avaliador.

Pré-condição:

A banca ter sido composta e ter os seus avaliadores atribuídos.

A avaliação é liberada para os avaliadores somente no dia da apresentação.

Sequência de Eventos:

1. Ator acessa a tela para realizar a avaliação.
2. O sistema apresenta as bancas que o usuário faz parte e que ainda estão com avaliação em aberto.
3. Ator escolhe a banca e o aluno para avaliar.
4. Sistema apresenta os itens que compõe a avaliação e para os quais deve ser informada a nota. Cada item apresentará o peso correspondente, se houver.
5. O ator lança os valores para as notas.
6. Sistema faz o cálculo da média e a apresenta.
7. Ator confirma os dados inserindo-os no banco de dados.
8. Sistema informa que os dados foram incluídos no banco de dados.

Pós-Condição:

Dados da avaliação inseridos no banco de dados.

Nome do fluxo alternativo (extensão)	Descrição
Linha 6: Avaliador quer alterar dados após a média ser apresentada.	6.1 Ao ser apresentada a média o avaliador tem a opção de salvar ou editar. O avaliador escolhe editar e retorna para o formulário de avaliação. 6.2 Retorna ao passo 6. A sequência do caso de uso prossegue com o fluxo normal.
Linha 7: Dados não são válidos.	7.1 No momento de salvar, o sistema faz a verificação e constata que há dados inválidos. É emitida mensagem 7.2 Retorna para o formulário de avaliação em estado de edição – passo 5.

Quadro 5 – Caso de uso realizar avaliação

O caso de uso manter cadastros é apresentado no Quadro 6.

Caso de uso:

Manter cadastros

Descrição:

Esse caso de uso se refere às operações de inclusão, exclusão, consulta e alteração dos cadastros necessários para a composição de uma banca e realização de avaliação. São os cadastros de avaliadores, alunos, itens de avaliação, grupos de itens, projetos, tipos de projetos, cursos, vincular alunos a cursos e papéis.

Evento Iniciador:

Necessidade de incluir, excluir, consultar ou alterar cadastros.

Atores:

Professor responsável pela atividade ou administrador.

Pré-condição:

Dados necessários disponíveis.

Sequência de Eventos:

1. Ator seleciona formulário no qual deseja realizar a operação.
2. Sistema apresenta o formulário.
3. Ator realiza a operação desejada: incluir, excluir, consultar, alterar.
4. Sistema verifica se os dados para a operação estão consistentes e realiza a operação.

Pós-Condição: Operação de inclusão, exclusão, consulta, alteração.	
Nome do fluxo alternativo (extensão)	Descrição
Linha 4: Dados não são válidos.	4.1 No momento de salvar, o sistema faz a verificação e constata que há dados inválidos. É emitida mensagem e retornado para o formulário de avaliação em estado de edição. Retorna ao passo 3 e o caso de uso prossegue com a sequência normal.
Linha 4: Exclusão de registro com dados de tabelas vinculadas ativos.	4.1 Se solicita exclusão de registro que possui tabelas vinculadas com dados ativos, é informada mensagem que não é possível realizar a exclusão. Sistema informa o usuário e permanece em estado anterior à realização da operação na Linha 4.

Quadro 6 – Caso de uso manter cadastros

O caso de uso compor avaliação é apresentado no Quadro 7.

Caso de uso: Compor avaliação **Descrição**: O professor responsável pela atividade que será avaliada faz a composição da avaliação. Uma avaliação é composta por uma série de itens. Esses itens são agrupados em categorias e a cada item pode estar associado a um peso. **Evento Iniciador**: Ator seleciona formulário para composição de uma avaliação. **Atores**: Professor responsável pelo tipo de atividade. **Pré-condição**: Itens de avaliação cadastrados. **Sequência de Eventos**: 1. Ator seleciona formulário para composição de avaliação. 2. Sistema apresenta formulário. 3. Ator seleciona grupo ou categoria. 4. Ator seleciona itens para compor a avaliação e pode atribuir pesos aos mesmos. Os itens 2 e 3 são repetidos até que o ator considere o modelo (*template*) da avaliação completo. 5. Sistema insere o modelo de avaliação no banco de dados. **Pós-Condição**: Modelo de avaliação cadastrado no sistema.	
Nome do fluxo alternativo (extensão)	Descrição
Linha5: Dados não são válidos.	5.1 No momento de salvar, o sistema faz a verificação e constata que há dados inválidos. 5.2 É emitida mensagem de dados inválidos. 5.3 É retornado para o formulário de avaliação em estado de edição – passo 4 (Linha 4)

Quadro 7 – Caso de uso compor avaliação

A descrição do caso de uso compor banca é apresentada no Quadro 8.

Caso de uso: Compor banca **Descrição**: O professor responsável pela atividade que será avaliada (estágio, proposta de trabalho de

conclusão de curso, proposta de trabalho de conclusão de curso, relatório de pesquisa, dentre outros) faz a composição da banca para avaliação de apresentação de trabalho. Compor uma banca significa associar os professores que farão a avaliação do respectivo trabalho.

Evento Iniciador:

Trabalho para ser avaliado por banca.

Atores:

Professor responsável.

Pré-condição:

Membros da banca e apresentação cadastradas, data e horário definidos.

Sequência de Eventos:

1. Ator abre formulário para composição de banca.
2. Ator seleciona a apresentação para a qual será composta a banca.
3. Ator seleciona os avaliadores que serão membros da banca e define data, local e horário de apresentação.
4. Sistema salva informações no banco e envia *email* aos membros da banca e autor(es) do trabalho informando-os da data, horário e local da apresentação.

Pós-Condição:

Banca composta.

Quadro 8 – Caso de uso compor banca

O Quadro 9 apresenta a descrição do caso de uso obter média.

Caso de uso:

Obter média

Descrição:

O professor responsável pela atividade que será avaliada (estágio, proposta de trabalho de conclusão de curso, proposta de trabalho de conclusão de curso, relatório de pesquisa, dentre outros), verifica a média calculada pelo sistema para o respectivo trabalho. Essa média será informada no sistema acadêmico para armazenamento e registro de diário de classe.

Evento Iniciador:

Professor solicita média (resultado de avaliação) de trabalho de aluno.

Atores:

Professor responsável.

Pré-condição:

Todos os membros da banca terem realizado a avaliação da respectiva banca.

Sequência de Eventos:

1. Ator abre formulário para verificação de médias. Ator indica período de tempo para verificar as médias.
2. Sistema obtém e lista as médias das apresentações no período.

Pós-Condição:

Listagem das médias apresentadas.

Nome do fluxo alternativo (extensão)	Descrição
Linha 2: Dados não são válidos.	2.1 O sistema indica que os dados são inválidos. 2.2 Os dados são solicitados novamente. 2.3 Retornar para o passo 1 (Linha 1).

Quadro 9 – Caso de uso obter média

A descrição do caso de uso emitir ata é apresentada no Quadro 10.

Caso de uso:

Emitir ata.

Descrição:

O professor responsável pela atividade que foi avaliada solicita a emissão da ata para ser anexada ao trabalho do aluno.

Evento Iniciador:

Professor seleciona aluno para emitir ata. **Atores**: Professor responsável. **Pré-condição**: Avaliação ter sido realizada. **Sequência de Eventos**: 1. Ator abre formulário para emissão de ata. Ator seleciona apresentação e aluno para emitir a ata. 2. Sistema emite a ata solicitada. **Pós-Condição**: Ata gerada.

Quadro 10 – Caso de uso emitir ata

O caso de uso manter usuários que é realizado pelo usuário administrador é apresentado no Quadro 11.

Caso de uso: Manter usuários **Descrição**: Realização do cadastro dos usuários do sistema. Os avaliadores terão acesso de avaliação somente às bancas das quais fazem parte. **Evento Iniciador**: Necessidade de incluir, excluir, consultar ou alterar usuários do sistema. **Atores**: Administrador. **Pré-condição**: Existência dos dados essenciais para realizar o cadastro. **Sequência de Eventos**: 1. Ator seleciona formulário no qual deseja realizar a operação. 2. Sistema apresenta o formulário. 3. Ator realiza a operação desejada: incluir, excluir, consultar e alterar usuários. 4. Sistema verifica se os dados para a operação estão consistentes e realiza a operação. **Pós-Condição**: Operação de inclusão, exclusão, consulta, alteração de usuários do sistema.	
Nome do fluxo alternativo (extensão)	Descrição
Linha 4: Dados não são válidos.	4.1 No momento de salvar, o sistema faz a verificação e constata que há dados inválidos. 4.2 É emitida mensagem informando que os dados são inválidos. 4.3 Retorna para o formulário de avaliação em estado de edição – passo 3 (Linha 3).

Quadro 11 – Caso de uso manter usuários

Na Figura 3 está o diagrama de classes de análise do sistema.

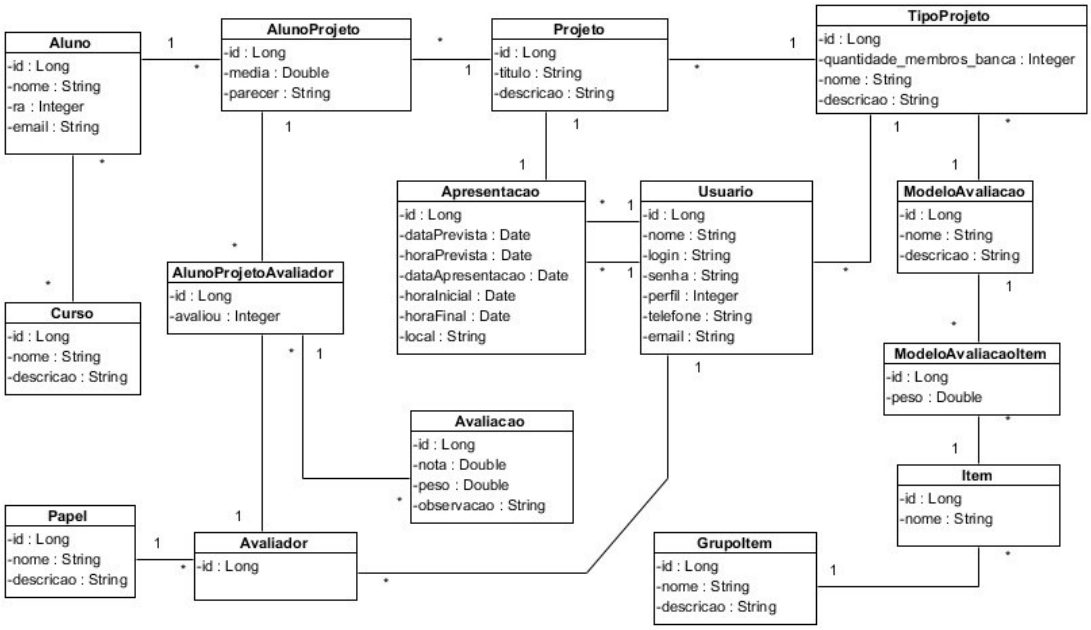

Figura 3 – Diagrama de classes de análise do sistema

As classes apresentadas no diagrama da Figura 3 estão documentadas a seguir. O Quadro 12 apresenta a classe Aluno. O atributo "id" das classes constantes na Figura 3 é necessário para o tratamento pelo Hibernate. Os métodos de inclusão, alteração, consulta e exclusão estão todos na classe *Data Access Object* (DAO). Foi criada uma classe genérica que recebe outra classe por parâmetro e realiza as operações na mesma. Assim, não é necessário criar um DAO para cada classe, pelo menos para as operações básicas (inserir, atualizar, excluir e listar todos os registros).

Identificação:	Aluno
Descrição:	Alunos que apresentam trabalhos perante bancas de avaliadores.
Requisitos:	RF03
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do aluno. ra (número): registro acadêmico do aluno. email (string): email de contato do aluno.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 12 – Descrição da classe Aluno

No Quadro 13 está a apresentação da classe Curso.

Identificação:	Curso
Descrição:	Cursos aos quais pertencem os alunos que apresentam trabalhos que serão avaliados por bancas.
Requisitos:	RF015
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do curso. descricao (string): complemento ao nome do curso.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 13 – Descrição da classe Curso

A classe usuários é apresentada no Quadro 14.

Identificação:	Usuario
Descrição:	Usuários com acesso ao sistema. Há três níveis de acesso ao sistema.
Requisitos:	RF016
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do aluno. login (string): identificação de acesso ao sistema. senha (string): para acesso ao sistema. administrador (número): indica as permissões de acesso ao sistema.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 14 – Descrição da classe Usuario

No Quadro 15 é apresentada a classe Projeto.

Identificação:	Projeto
Descrição:	Projetos é a denominação genérica para os trabalhos apresentados.
Requisitos:	RF02
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. título (string): nome do trabalho a ser apresentado. descricao (string): complemento ao nome do curso.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 15 – Descrição da classe Projeto

No Quadro 16 é apresentada a descrição da classe que vincula alunos a projetos.

Identificação:	AlunoProjeto
Descrição:	Vincular alunos a projetos.
Requisitos:	RF03, RF04.
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. mediaFinal (número): média obtida pelo alunos, resultante da avaliação de todos os membros da banca. parecer (string): resultado final (aprovado, por exemplo). Esse campo pode ser utilizado quando não a avaliação não é quantitativa.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 16 – Descrição da classe AlunoProjeto

No Quadro 17 está a apresentação da classe ModeloAvaliacao.

Identificação:	ModeloAvaliacao
Descrição:	As avaliações disponibilizadas aos membros das bancas.
Requisitos:	RF16
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do modelo de avaliação. descricao (string): complemento ao nome do modelo de avaliação.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 17 – Descrição da classe ModeloAvaliaçao

No Quadro 18 está a apresentação da classe Item.

Identificação:	Item
Descrição:	Itens que compõem uma avaliação
Requisitos:	RF06, RF07
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do item que compõem um modelo de avaliação.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 18 – Descrição da classe Item

No Quadro 19 está a apresentação da classe ModeloAvaliacaoItem.

Identificação:	ModeloAvaliacaoItem
Descrição:	Um modelo de avaliação
Requisitos:	RF17
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. peso (string): peso do item que compõe o modelo de avaliação.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 19 – Descrição da classe ModeloAvaliaçaoItem

No Quadro 20 está a apresentação da classe GrupoItem.

Identificação:	GrupoItem
Descrição:	Os grupos de itens de avaliação.
Requisitos:	RF07
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do grupo de itens que compõem um modelo de avaliação. descricao (string): complemento ao nome do grupo de itens.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 20 – Descrição da classe GrupoItem

No Quadro 21 está a apresentação da classe TipoProjeto.

Identificação:	TipoProjeto
Descrição:	Um projeto é de um tipo (estágio, TCC, proposta, dentre outros). O tipo define a quantidade de membros da banca e permite fornecer acesso às bancas somente ao professor responsável por aquele tipo de projeto.
Requisitos:	RF12
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. quantidade_membros_banca (número): quantidade de membros banca. nome (string): nome do tipo de projeto ou atividade a ser avaliada. descricao (string): complemento ao nome do tipo de projeto.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 21 – Descrição da classe TipoProjeto

No Quadro 22 é apresentada a descrição da classe responsável pelos dados e métodos relacionados à apresentação de um trabalho.

Identificação:	Apresentação
Descrição:	Classe responsável pelos dados e métodos relacionados à apresentação de um trabalho acadêmico perante uma banca de avaliadores.
Requisitos:	RF01
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. dataPrevista (data): data previamente agendada para a apresentação. dataApresentacao (data): data na qual a apresentação foi realizada. horaInicial (hora): horário de inicio da apresentação. horaFinal (hora): horário de fim da apresentação. localApresentacao (string): ambiente de apresentação do trabalho.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 22 – Descrição da classe Apresentacao

No Quadro 23 é apresentada a descrição da classe Avaliador.

Identificação:	Avaliador
Descrição:	Pessoas participantes das bancas de avaliação, geralmente professores.
Requisitos:	RF05
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do aluno. telefone (string): número de telefone. email (string): endereço eletrônico.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 23 – Descrição da classe Avaliador

No Quadro 24 está a apresentação da classe Avaliacao.

Identificação:	Avaliacao
Descrição:	Uma avaliação é composta por um conjunto de itens aos quais será atribuída nota.
Requisitos:	RF10
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. notaItem (número): nota atribuída ao respectivo item.
Métodos:	void adiciona(T t); void atualiza(T t);

	void remove(T t); List<T> listaTodos(); T buscaPorId(Long id);

Quadro 24 – Descrição da classe Avaliacao

No Quadro 25 está a apresentação da classe Papel.

Identificação:	Papel
Descrição:	Um participante de uma banca de avaliação realiza papéis como: orientador, supervisor, convidado.
Requisitos:	RF18
Atributos:	id (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): nome do papel. descricao (string): complemento ao nome do papel.

Métodos:	void adiciona(T t);
	void atualiza(T t);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);

Quadro 25 – Descrição da classe Papel

A Figura 4 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação.

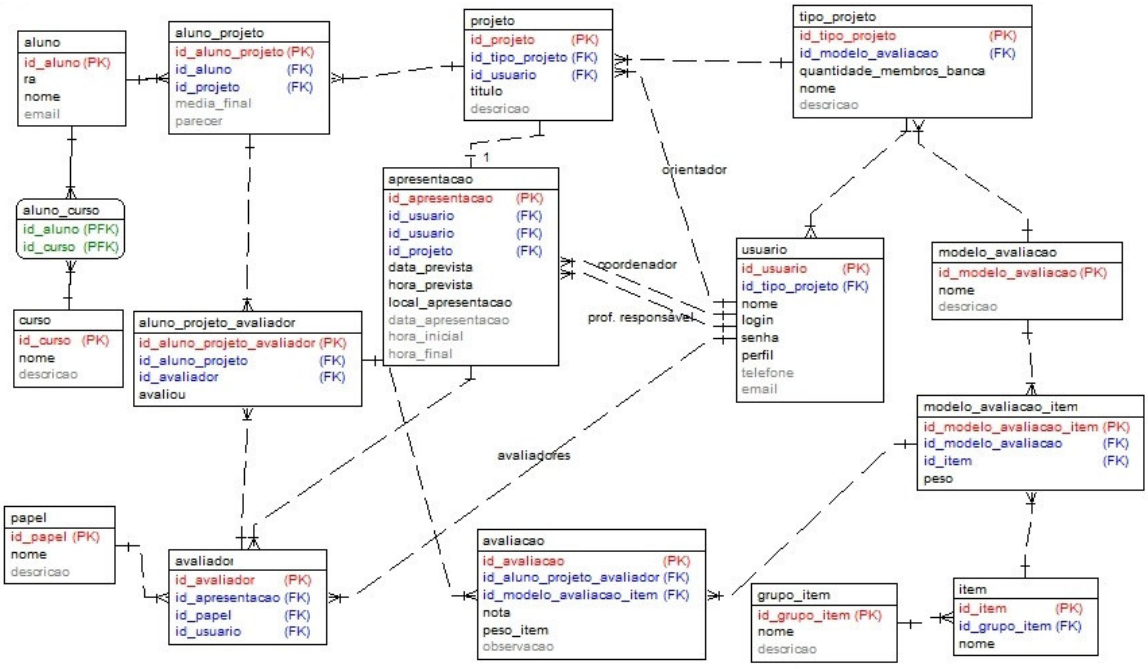

Figura 4 – Diagrama de entidades e relacionamentos do banco de dados

No Quadro 26 estão os campos da tabela de alunos. Um aluno realizará uma apresentação de um projeto que será objeto de avaliação. Um aluno pertence a um curso.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
id_aluno	Numérico	Não	Sim	Não	
ra	Numérico	Não	Não	Não	
nome	Texto	Não	Não	Não	
email	Texto	Sim	Não	Não	

Quadro 26 – Campos da tabela aluno

O Quadro 27 apresenta os campos da tabela de cursos.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
id_curso	Numérico	Não	Sim	Não	
nome	Texto	Não	Não	Não	
descricao	Texto	Sim	Não	Não	

Quadro 27 – Campos da tabela curso

Alunos estão vinculados a um curso. O Quadro 28 apresenta os campos da tabela que vincula alunos a cursos.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
id_aluno	Numérico	Não	Sim	Sim	Vem da tabela aluno
id_curso	Numérico	Não	Sim	Sim	Vem da tabela curso

Quadro 28 – Campos da tabela aluno_curso

Os campos da tabela usuários estão no Quadro 29. Usuários são as pessoas que possuem acesso ao sistema. Os avaliadores das bancas, os professores responsáveis pelos tipos de projeto, coordenadores e orientadores são provenientes dessa tabela. Avaliadores são membros que compõem as bancas. O professor responsável pelas atividades avaliadas como banca (estágio e trabalho de conclusão de curso, por exemplo), o coordenador do curso e o orientador são também oriundos dessa tabela. O campo perfil permite definir o tipo de acesso (administrador, avaliador, responsável) que o usuário terá. O avaliador ao acessar o sistema somente visualizará as bancas das quais ele é membro. A composição e a atribuição das bancas são realizadas pelo professor responsável pelo respectivo tipo de projeto.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_usuario	Numérico	Não	Sim	Não	
id_tipo_projeto	Numérico	Não	Não	Sim	Da tabela tipo_projeto
nome	Texto	Não	Não	Não	
telefone	Texto	Sim	Não	Não	
email	Texto	Sim	Não	Não	
login	Texto	Não	Não	Não	
senha	Texto	Não	Não	Não	
perfil	Texto	Sim	Não	Não	

Quadro 29 – Campos da tabela usuario

O Quadro 30 apresenta os campos da tabela papeis. Os papéis são exercidos pelos participantes das bancas e que devem constar na ata de realização das bancas. Entre os papéis estão: professor orientador, coordenador de curso, membro de banca, supervisor e responsável pela atividade (estágio e trabalho de conclusão de curso, por exemplo).

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_papel	Numérico	Não	Sim	Não	
nome	Texto	Não	Não	Não	
descricao	Texto	Sim	Não	Não	

Quadro 30 – Campos da tabela papel

Um projeto representa a atividade realizada pelo aluno e que será apresentada perante a banca. Os projetos são de estágio, trabalho de conclusão de curso, de pesquisa, de proposta de trabalho de conclusão de curso, dentre outros. No Quadro 31 estão os campos da tabela de projetos.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_projeto	Numérico	Não	Sim	Não	
id_tipo_projeto	Numérico	Não	Não	Sim	Vem da tabela tipo_projeto
id_usuario	Numérico	Não	Não	Sim	Vem da tabela usuarios. É o professor orientador
nome	Texto	Não	Não	Não	
descricao	Texto	Não	Não	Não	

Quadro 31 – Campos da tabela projeto

O resultado da avaliação, como a média obtida das notas atribuídas pelos avaliadores, e um parecer da banca (opcional) é armazenado para que do mesmo possa ser gerada a listagem para lançamento no sistema acadêmico. Os campos da tabela que armazena os dados de projetos e alunos estão apresentados no Quadro 32.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_aluno_projeto	Numérico	Não	Sim	Não	
id_aluno	Numérico	Não	Não	Sim	Vem da tabela aluno
id_projeto	Numérico	Não	Não	Sim	Vem da tabela projeto
media_final	Numérico	Sim	Não	Não	
parecer	Texto	Sim	Não	Não	

Quadro 32 – Campos da tabela aluno_projeto

Os projetos são categorizados por tipo. O tipo de projeto define a composição da banca em termos de quantidade de membros e o modelo de avaliação que será utilizado. No Quadro 33 estão os campos para armazenar os dados referentes ao tipo de projeto.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_tipo_projeto	Numérico	Não	Sim	Não	
id_modelo_ avaliacao	Numérico	Não	Não	Sim	Vem da tabela modelo_avaliacao
nome	Texto	Não	Não	Não	
descricao	Texto	Sim	Não	Não	
quantidade_ membros_banca	Numérico	Não	Não	Não	

Quadro 33 – Campos da tabela de tipo_projeto

Os modelos de avaliação são utilizados pelos membros das bancas para realizar a avaliação. O Quadro 34 apresenta os campos da tabela de tipos de modelos de avaliação.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_modelo_ avaliacao	Numérico	Não	Sim	Não	
nome	Texto	Não	Não	Não	
descricao	Texto	Sim	Não	Não	

Quadro 34 – Campos da tabela modelo_avaliacao

Uma avaliação é composta por diversos itens que são agrupados. No Quadro 35 estão os campos da tabela que armazena os dados de grupos de itens.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_grupo_item	Numérico	Não	Sim	Não	
nome	Texto	Não	Não	Não	
descricao	Texto	Sim	Não	Não	

Quadro 35 – Campos da tabela grupo_item

Os itens que compõem uma avaliação são apresentados no Quadro 36.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_item	Numérico	Não	Sim	Não	
id_grupo_item	Numérico	Não	Não	Sim	Vem da tabela grupo_item
nome	Texto	Não	Não	Não	

Quadro 36 – Campos da tabela item

O Quadro 37 apresenta os campos da tabela que armazena a composição de uma avaliação. Uma avaliação é composta por um conjunto de itens.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_modelo_avaliacao_item	Numérico	Não	Sim	Não	
id_modelo_ avaliacao	Numérico	Não	Não	Sim	Vem da tabela modelo_avaliacao
id_item	Numérico	Não	Não	Sim	Vem da tabela item
peso	Numérico	Não	Não	Não	

Quadro 37 – Campos da tabela modelo_avaliacao_item

Uma banca ocorre quando alunos apresentam trabalhos para serem avaliados. O Quadro 38 apresenta os campos da tabela da avaliacao.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_avaliacao	Numérico	Não	Sim	Não	
id_aluno_projeto_avaliador	Numérico	Não	Não	Sim	Vem da tabela aluno_projeto
id_modelo_avaliacao_item	Numérico	Não	Não	Sim	Vem da tabela modelo_avaliacao
nota	Numérico	Sim	Não	Não	Média calculada pela nota dos itens e respectivos pesos
peso_item	Numérico	Sim	Não	Não	
observacao	Numérico	Sim	Não	Não	

Quadro 38 – Campos da tabela avaliacao

A avaliação ocorre quando da apresentação de um projeto perante uma banca. No Quadro 39 estão apresentados os campos da tabela da apresentacao.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_apresentacao	Numérico	Não	Sim	Não	
id_projeto	Numérico	Não	Não	Sim	Vem da tabela projeto
id_usuario	Numérico	Não	Não	Sim	Vem da tabela avaliador. Coordenador do curso
id_usuario	Numérico	Não	Não	Sim	Vem da tabela avaliador. Professor orientador
data_prevista	Data	Não	Não	Não	
hora_prevista	Hora	Não	Não	Não	
data_apresentacao	Data	Sim	Não	Não	
hora_inicial	Hora	Sim	Não	Não	
hora_final	Hora	Sim	Não	Não	
local_apresentacao	Texto	Não	Não	Não	

Quadro 39 – Campos da tabela apresentacao

O Quadro 40 apresenta os campos da tabela de avaliador. Avaliador é a pessoa que participa de uma banca e realiza a avaliação do trabalho apresentado.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_avaliador	Numérico	Não	Sim	Não	Vem da tabela usuarios
id_apresentacao	Numérico	Não	Não	Sim	Vem da tabela apresentacao
id_papel	Numérico	Não	Não	Sim	Vem da tabela papel
id_usuario	Numérico	Não	Não	Sim	Vem da tabela usuario

Quadro 40 – Campos da tabela avaliador

Quadro 41 apresenta os campos da tabela que relaciona projetos, alunos e avaliadores. Um projeto pode ser composto por mais de um aluno, mas cada aluno é avaliado individualmente. Essa tabela tem o objetivo de auxiliar no controle se todos os avaliadores de uma determinada banca já realizaram a avaliação.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_aluno_projeto_avaliador	Numérico	Não	Sim	Não	Vem da tabela usuarios
id_aluno_projeto	Numérico	Não	Não	Sim	Vem da tabela aluno_projeto
id_avaliador	Numérico	Não	Não	Sim	Vem da tabela usuario
avaliou	Numérico	Sim	Não	Não	

Quadro 41 – Campos da tabela aluno_projeto_avaliador

4.3 APRESENTAÇÃO DO SISTEMA

A tela inicial do sistema é a tela de *login* apresentada na Figura 5. Por meio dessa tela é realizado o acesso ao sistema.

Figura 5 – Tela de login

Ao tentar efetuar o acesso, é feita uma validação do usuário e da senha informados. Caso essas informações sejam inválidas, o usuário é informado por meio de uma mensagem apresentada na tela, como mostra a Figura 6.

Figura 6 - Login inválido

Ao efetuar o acesso com sucesso (*login* e senha válidos), o usuário é encaminhado para a tela de boas-vindas do sistema que é apresentada na Figura 7. O menu na lateral esquerda apresenta as opções (funcionalidades) para o perfil do usuário que está logado.

Figura 7 – Tela de boas-vindas

O leiaute do sistema é composto por três setores: o setor superior contém a logo da UTFPR, o setor lateral esquerdo contém o menu de navegação e o setor central contém o conteúdo da página que está sendo navegada. Os dois primeiros setores são estáticos. A Figura 8 apresenta os três setores nos quais a página foi organizada.

Figura 8 – Leiaute da página do sistema

O setor lateral esquerdo pode ser minimizado, se necessário, deixando um espaço maior para o usuário trabalhar, como mostra a Figura 9.

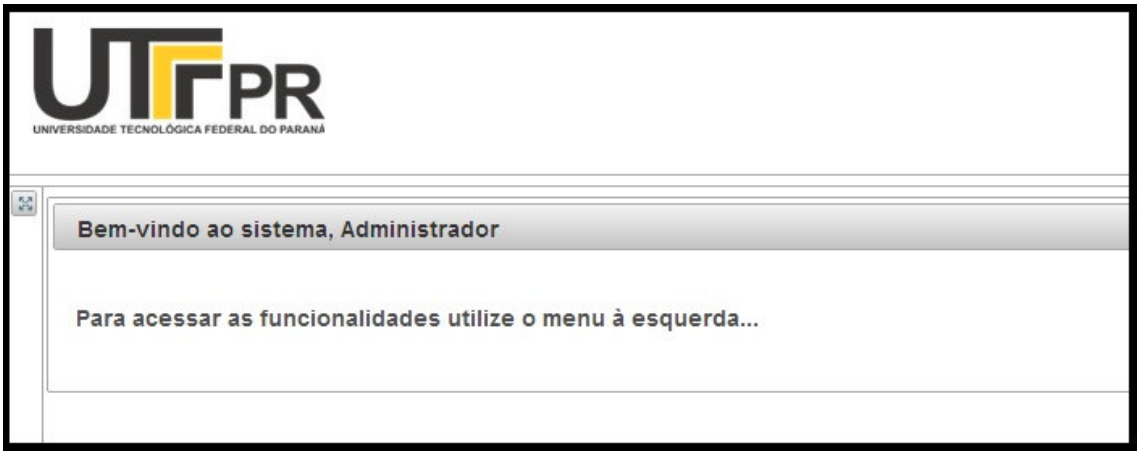

Figura 9 – Menus minimizados

O usuário logado poderá alterar seus dados e/ou sua senha. Para isso basta clicar sobre o seu *login*, localizado no canto superior direito da tela, ao lado do *link* "Sair" que é responsável por efetuar o *logoff*. Assim, será apresentado um menu com duas opções: uma para alterar os dados e outra para alterar a senha, como mostra a Figura 10.

Figura 10 – Menu para alterar dados do usuário logado

Na Figura 11 é apresentada a tela de alteração dos dados do usuário logado.

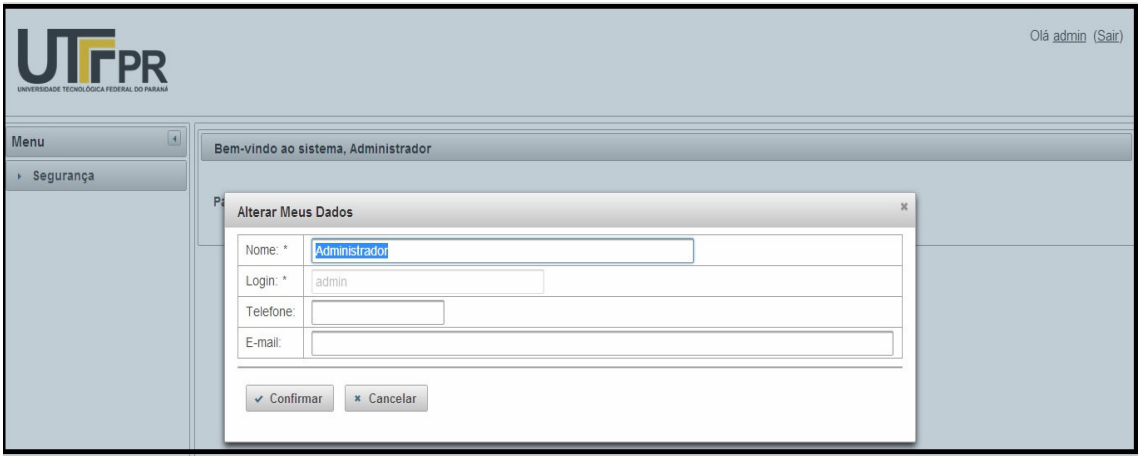

Figura 11 – Alterar dados do usuário logado

Por meio da Figura 11 é possível verificar que o campo para alterar o *login* está desabilitado. Isso ocorre apenas se o usuário logado é o usuário "admin", pois ele é o usuário padrão do sistema e seu *login* não poderá ser alterado. Na Figura 12 é apresentada a tela de alteração da senha do usuário logado.

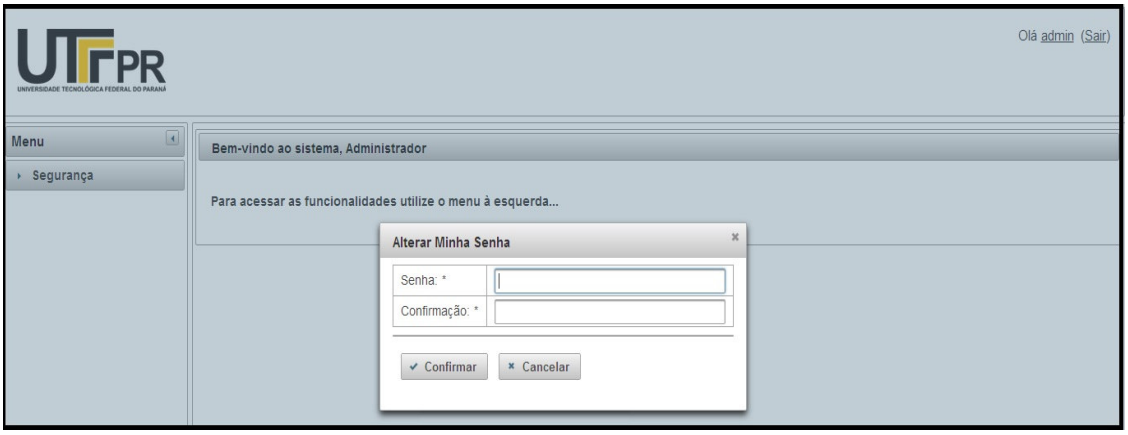

Figura 12 – Alterar senha do usuário logado.

A Figura 13 mostra apenas o setor central, com o conteúdo da página de cursos. Nessa área o usuário visualiza os cursos já cadastrados e também pode realizar as operações básicas de cadastro como inclusão, alteração e exclusão.

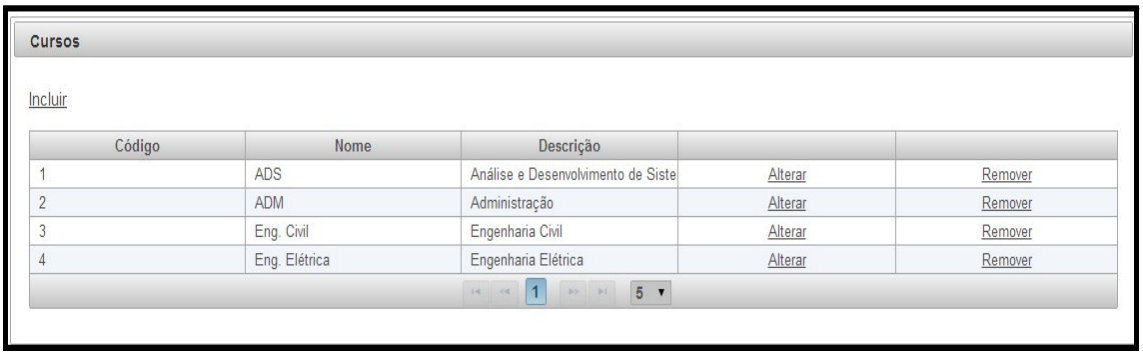

Figura 13 – Lista de cursos

Ao clicar no *link* "Incluir" (Figura 13) o sistema ocultará a lista de cursos cadastrados e mostrará o formulário com os campos necessários para a inclusão de um novo curso, juntamente com os botões para confirmar ou cancelar, que ocorre por meio de uma requisição Ajax, como mostra a Figura 14.

Figura 14 – Formulário para incluir cursos

Se o usuário clicar no botão "Confirmar" sem informar os campos obrigatórios o sistema fará uma validação e apresentará uma mensagem para o usuário informar o campo obrigatório antes de salvar, como mostra a Figura 15.

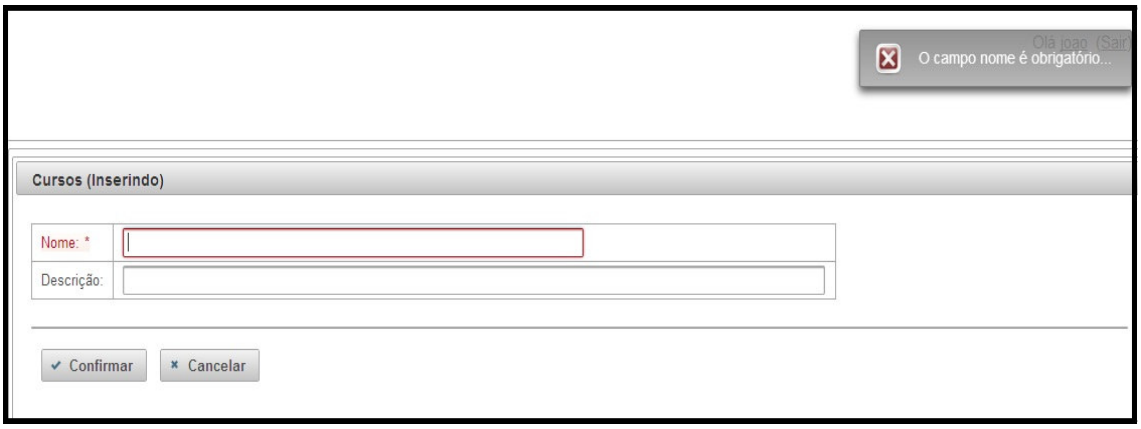

Figura 15 – Exemplo de validação de campos obrigatórios

Se os campos forem válidos, o sistema gravará o novo curso, ocultará o formulário de cadastro, mostrará novamente a lista de cursos cadastrados atualizada e apresentará uma mensagem de sucesso, como mostra a Figura 16.

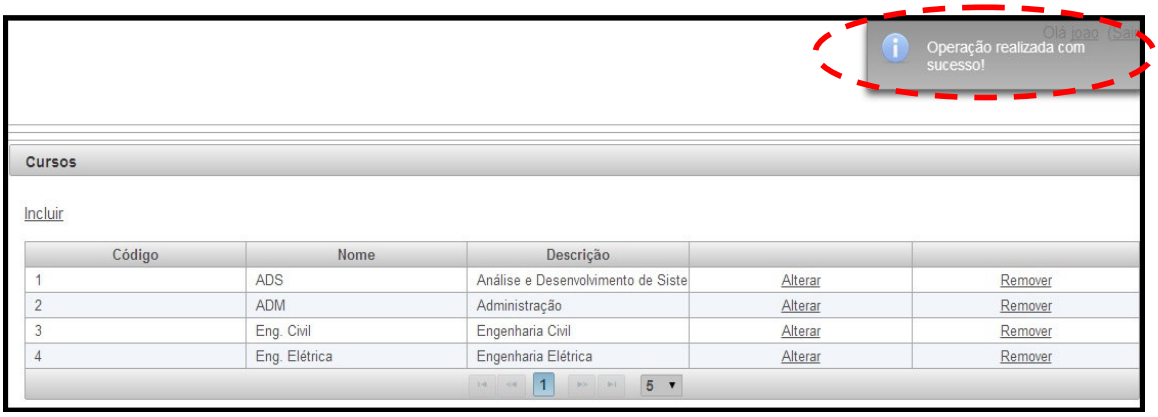

Figura 16 – Confirmação da operação realizada

A operação de alteração é semelhante à de inclusão, sendo que, para alterar um registro desejado o usuário deve clicar no *link* "Alterar" da linha referente ao curso desejado. Desta forma, o sistema fará o mesmo procedimento utilizado na inclusão, mas já trará o formulário preenchido com os dados do curso selecionado, como mostra a Figura 17. O processo de validação também é o mesmo, para salvar as alterações, os campos obrigatórios devem estar preenchidos.

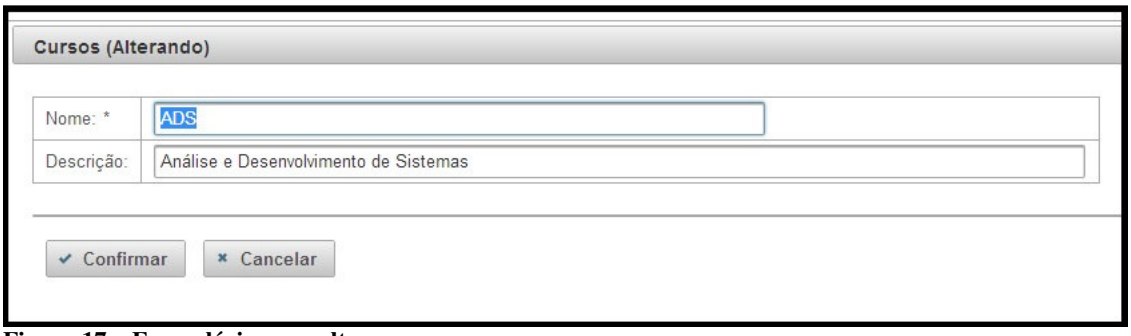

Figura 17 – Formulário para alterar cursos

Para realizar a operação de exclusão, o usuário deve clicar no *link* "Remover" da linha referente ao curso desejado. O sistema apresentará uma mensagem de confirmação informando o usuário que a operação não poderá ser desfeita, como mostra a Figura 18.

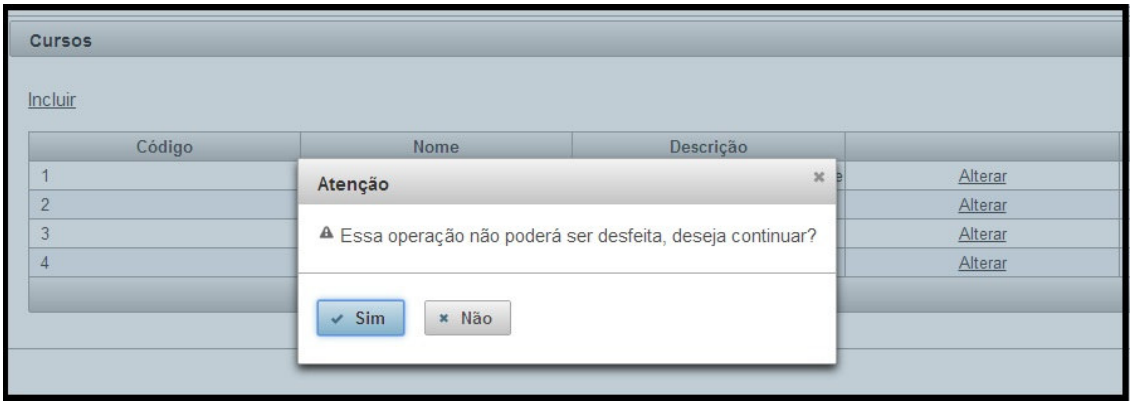

Figura 18 – Confirmação para remover curso

Se o usuário confirmar a exclusão, o sistema removerá o curso desejado, atualizará a tabela de cursos cadastrados e apresentará uma mensagem de sucesso igual à mensagem apresentada na Figura 16.

Se o registro a ser removido estiver vinculado a outro registro, de outra entidade, o sistema não permitirá a exclusão e informará o usuário por meio de uma mensagem, como mostra a Figura 19.

Figura 19 – Registro vinculado a outro

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, que não possuem operações específicas, é o mesmo.

A entidade de projetos, por exemplo, possui um formulário mais específico, como mostra a Figura 20. No momento da inclusão ou alteração, o usuário deve informar os alunos do projeto, o que caracteriza esse formulário como mestre/detalhe.

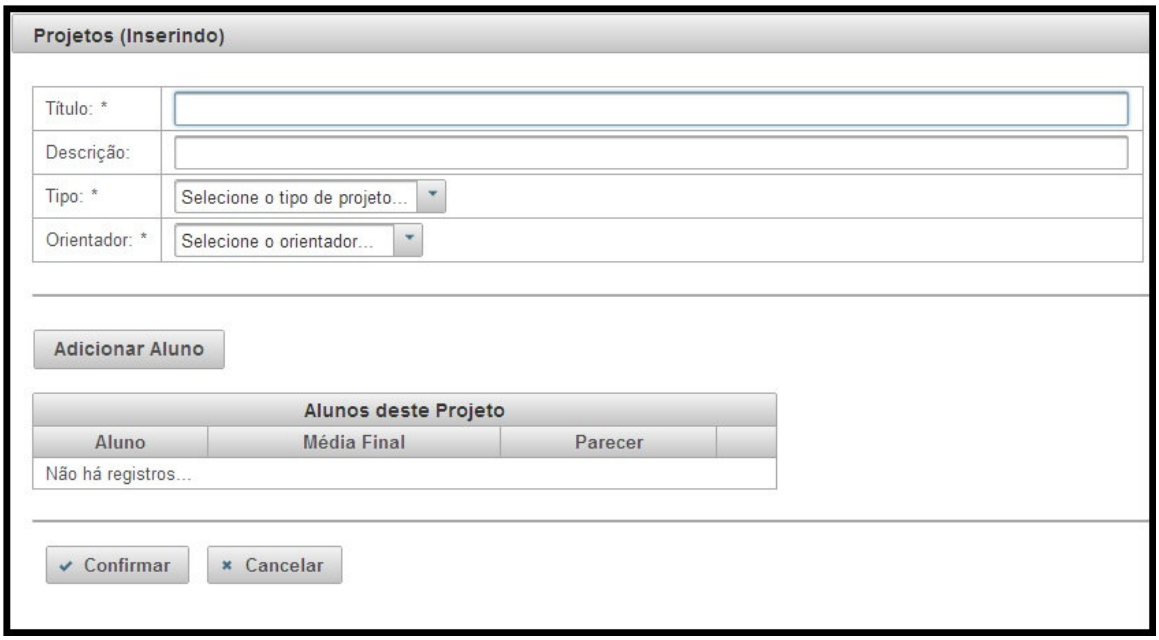

Figura 20 – Formulário de inclusão de projetos (mestre/detalhe)

Ao clicar no botão “Adicionar Aluno” é apresentada a tela para o usuário selecionar o aluno que pertence ao projeto, como mostra a Figura 21.

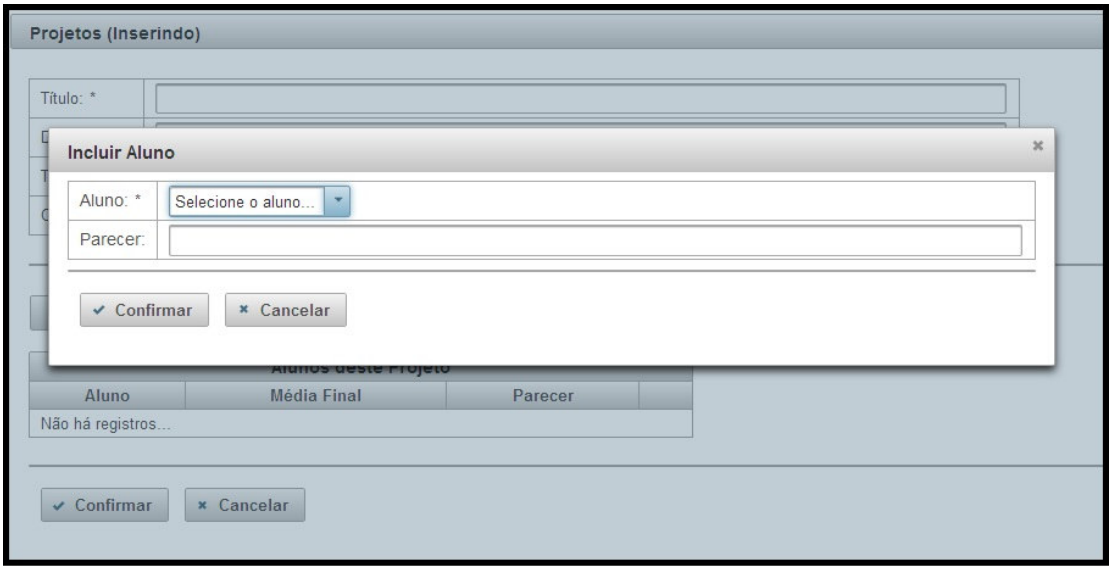

Figura 21 – Adicionar aluno ao projeto

O processo de validação, salvamento e exclusão da entidade de projetos, é igual as demais entidades do sistema.

Outra entidade que possui um processo mais específico é a entidade de apresentações. Nessa entidade o usuário compõe as apresentações a serem realizadas. A Figura 22 mostra o formulário de composição da apresentação.

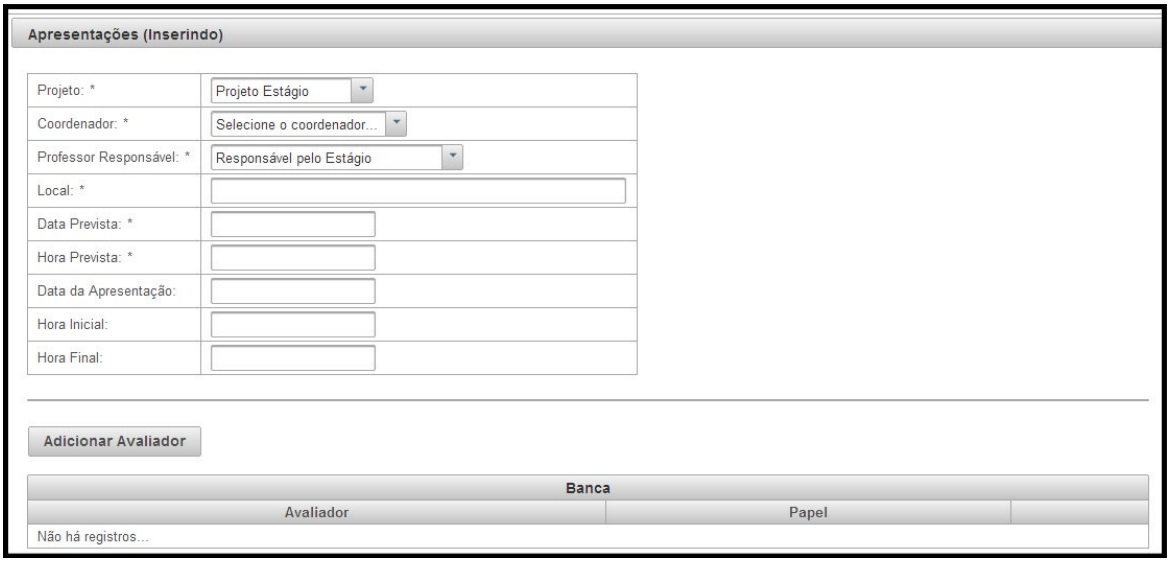

Figura 22 – Formulário para composição da apresentação

Nesse formulário, o usuário também compõe a banca avaliadora da apresentação. Ao selecionar o projeto, o sistema localizará o professor responsável pelo mesmo tipo de projeto e irá adicioná-lo à apresentação. Da mesma forma como no formulário de projetos, ao clicar no botão "Adicionar Avaliador", será apresentada a tela para o usuário selecionar e adicionar um avaliador à banca como mostra a Figura 23.

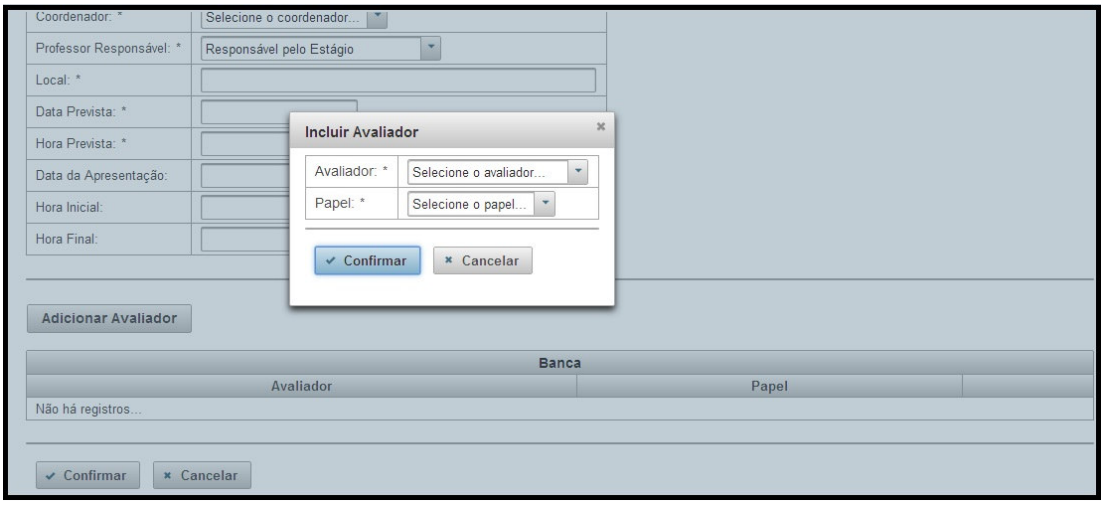

Figura 23 – Adicionar avaliador à apresentação

Após a apresentação estar composta os membros da banca poderão avaliá-la a partir do momento em que a data prevista da mesma for menor ou igual que a data atual. A Figura 24 mostra a lista de avaliações pendentes de um usuário avaliador.

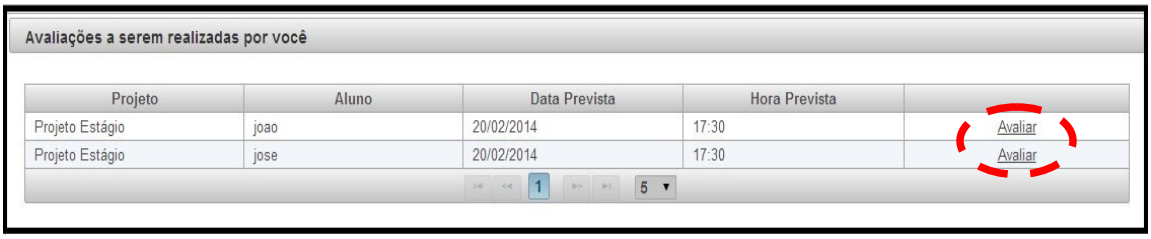

Figura 24 – Avaliações pendentes

Ao clicar no *link* "Avaliar" (área circulada na Figura 24), o sistema apresentará o formulário para o usuário realizar a avaliação da apresentação por aluno como mostra a Figura 25.

Figura 25 – Avaliar apresentação por aluno

Ao carregar o formulário de avaliação, o sistema buscará os itens de avaliação referentes ao modelo de avaliação e ao tipo de projeto de projeto avaliado. Ao clicar no botão "Adicionar Avaliação" o sistema apresentará a tela para o usuário lançar a nota no item da sequência como mostra a Figura 26.

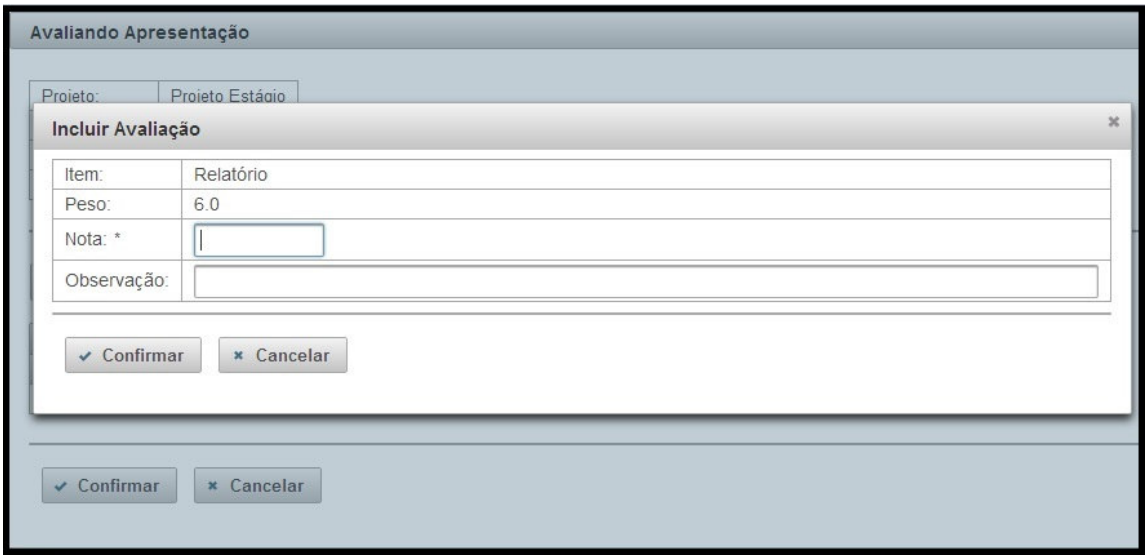

Figura 26 – Adicionar avaliação para a apresentação

Ao terminar a sequência dos itens da determinada apresentação, o sistema desabilitará o botão "Adicionar Avaliação" para que o usuário encerre a avaliação, como mostra a Figura 27.

Figura 27 – Encerrar a avaliação

Após todos os avaliadores de uma referida banca realizarem as avaliações, o sistema calculará a nota final do aluno e gravará no banco de dados. Depois desse procedimento realizado, os professores responsáveis poderão visualizar as avaliações por meio da lista de apresentações. A Figura 28 mostra a lista de apresentações que contém a opção para o usuário visualizar as avaliações.

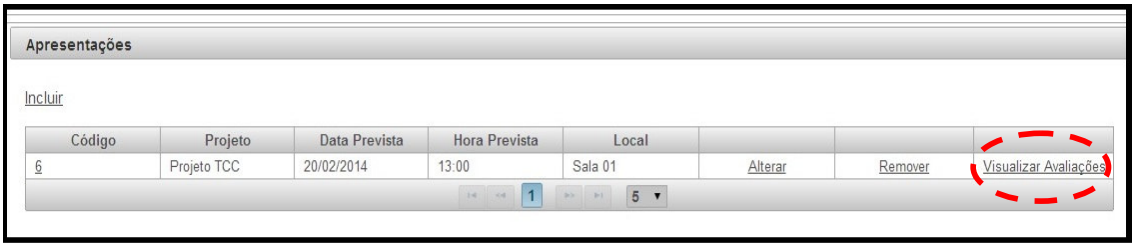

Figura 28 – Lista de apresentações

Ao clicar no *link* "Visualizar Avaliações" (área circulada da Figura 28) o sistema apresentará a tela que contém todas as avaliações da apresentação selecionada, como mostra a Figura 29. Dessa forma, o usuário consegue visualizar detalhadamente as avaliações realizadas e se existe alguma avaliação pendente.

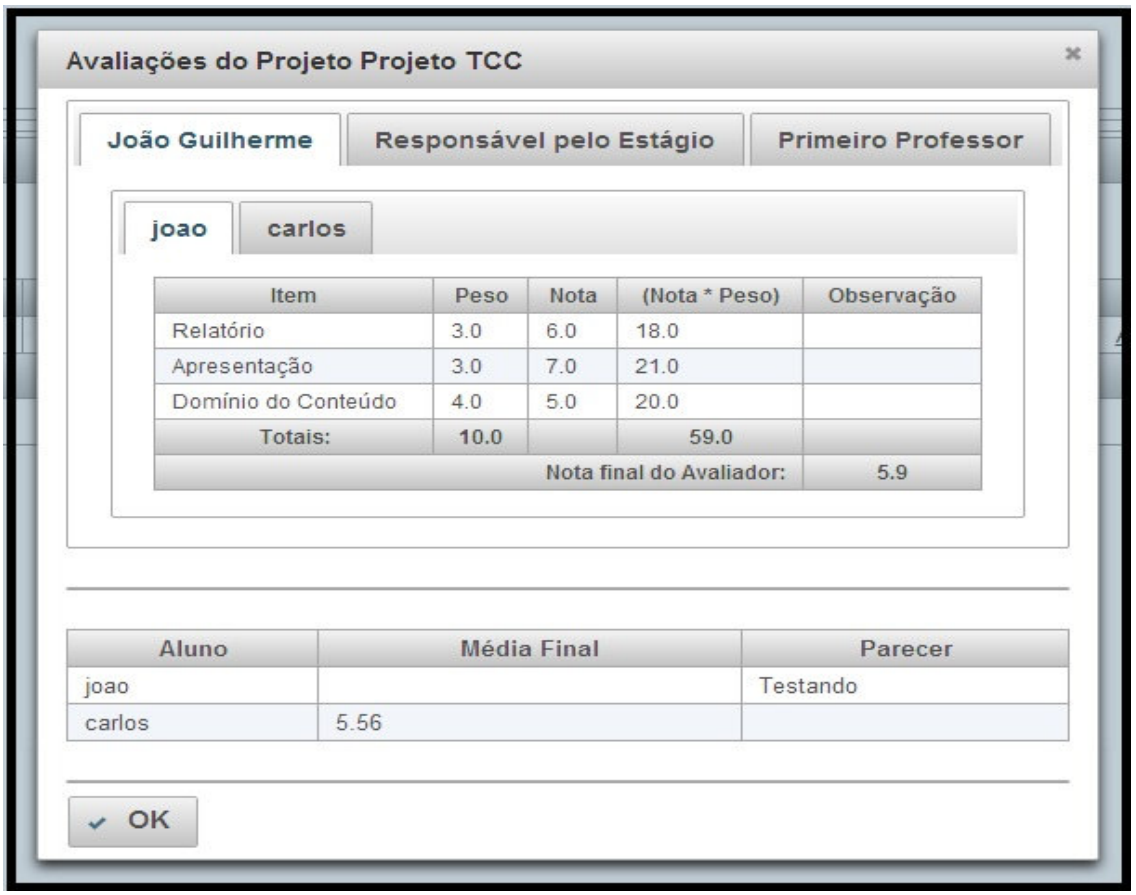

Figura 29 – Visualizar todas as avaliações da apresentação

4.4 IMPLEMENTAÇÃO DO SISTEMA

Como apresentado na Seção 4.3, o leiaute do sistema é composto por três setores, isso é feito por meio do uso do componente “layout” do Primefaces. Esse componente pode dividir a página em até cinco setores, sem a necessidade de utilizar as *tags* “div” do HTML e um arquivo CSS. Esses setores são chamados de “layoutUnits” e podem ter as seguintes posições: “north”, “south”, “east”, “west” e “center”.

Para facilitar o desenvolvimento, foi utilizada uma página modelo, chamada “_template.xhtml” (Listagem 1). Essa página contém o leiaute básico do sistema, que será utilizado por todas as outras páginas.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<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">

<h:head>
      <style type="text/css">
* {
      font-size: 98%;
}
</style>
</h:head>

<h:body>
      <p:layout fullPage="true">
            <p:layoutUnit position="north" size="120">
                  <h:graphicImage value="/resources/img/logo.png" style="margin:
1%;" />
            </p:layoutUnit>
            <p:layoutUnit position="west" size="250" collapsible="true"
                  header="Menu">
                  <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"
                        width="500">
                        <p:commandButton value="Sim" type="button"
                              styleClass="ui-confirmdialog-yes" icon="ui-icon-
check" />
                        <p:commandButton value="Não" type="button"
                              styleClass="ui-confirmdialog-no" icon="ui-icon-
close" />
                  </p:confirmDialog>
                  <ui:insert name="conteudo">
```

```
                        <!-- Aqui ficará o conteúdo preenchido pelas outras
páginas -->
                    </ui:insert>
                </p:layoutUnit>
        </p:layout>
</h:body>

</html>
```

Listagem 1 – _template.xhtml

Como visto na Listagem 1, foi utilizado apenas o componente "layout" do Primefaces, com três "layoutUnits" posicionados em "north", "west" e "center". A primeira *unit* contém o componente "graphicImage" do *Java Server Faces* (JSF), responsável por renderizar imagens. É ele quem apresenta a logo da UTFPR. A segunda *unit* faz a inclusão da página "menu.xhtml" por meio da *tag* "include" do Facelet. A terceira *unit* contém o componente "growl" do Primefaces, responsável por renderizar as mensagens que devem ser visualizadas pelo usuário, o componente "confirmDialog" também do Primefaces, responsável por renderizar mensagens de confirmação ao usuário e a *tag* "insert" do Facelet que serve para definir um "espaço" que será utilizado pelas outras páginas para atualizar o conteúdo da *unit*. Também foi utilizado um *script* do tipo CSS interno para alterar o tamanho da fonte de todos os componentes. Como essa página é a de modelo, esse *script* afetará todas as outras páginas.

O menu principal do sistema foi desenvolvido em uma página chamada "menu.xhtml" (apresentada na Listagem 2). Essa página usa a *tag* "fragment" do Facelet. Essa página é um fragmento que pode ser incluído em outras páginas. Também foi utilizado o componente "panelMenu" do Primefaces. Esse componente pode conter "submenus" e "menuitems" como mostra a Listagem 2.

```
<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">

        <h:form>
                <p:panelMenu>
                        <p:submenu label="Cadastros">
                                <p:menuitem value="Alunos" action="aluno?faces-
redirect=true" />
                                <p:menuitem value="Cursos" action="curso?faces-
redirect=true" />
                        </p:submenu>
                </p:panelMenu>
        </h:form>
</ui:fragment>
```

Listagem 2 – menu.xhtml

A página responsável pelo gerenciamento de cursos é chamada de “curso.xhtml” e utiliza as *tags* “composition” e “define” do Facelet. Quando usamos a *tag* “composition” dizemos que a página será a composição de uma página modelo, no nosso caso “_template.xhtml”, quando utilizamos a *tag* “define” devemos utilizar no atributo “name” o mesmo valor do atributo “name” da tag “insert” que pretendemos usar. A Listagem 3 mostra a página de cursos com o atributo “name” da *tag* “define” que possui o mesmo valor que o atributo “name” da *tag* “insert” utilizada na página modelo. Dessa forma, todo o conteúdo da página de cursos será inserido no “espaço” utilizado pela *tag* “insert” da página modelo.

```
<ui:composition template="/templates/_template.xhtml"
	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">

	<ui:define name="conteudo">
		<p:panel id="painelLista" header="Cursos cadastrados"
			visible="#{!cursoBean.estaCadastrando()}">
			<br />
			<h1>
				<p:commandLink value="Novo Curso"
					actionListener="#{cursoBean.prepararIncluir()}"
process="@this"
					update="painelLista painelCadastro" />
			</h1>
			<br />
			<p:dataTable id="tabelaPrincipal" value="#{cursoBean.cursos}"
				var="curso" paginator="true" rows="5"
				paginatorTemplate="{FirstPageLink} {PreviousPageLink}
{PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
				rowsPerPageTemplate="5,10,15"
				emptyMessage="Não há cursos cadastrados...">
				<p:column headerText="Código">
					<h:outputText value="#{curso.id}" />
				</p:column>
				<p:column headerText="Nome">
					<p:commandLink value="#{curso.nome}"

	action="#{cursoBean.prepararAtualizar(curso.id)}" process="@this"
						update=":painelLista :painelCadastro"
						title="Atualizar" />
				</p:column>
				<p:column headerText="Descrição">
					<h:outputText value="#{curso.descricao}" />
				</p:column>
				<p:column>
					<p:commandLink value="Remover"
						action="#{cursoBean.remover(curso.id)}"
process="@this"
						update="tabelaPrincipal">
						<p:confirm header="Atenção"
							message="Essa operação não poderá ser
desfeita, deseja continuar?"
```

```
                                                icon="ui-icon-alert" />
                                    </p:commandLink>
                              </p:column>
                        </p:dataTable>
                        <br />
                  </p:panel>
                  <p:panel id="painelCadastro"
                        header="#{cursoBean.tituloPainelCadastro}"
                        visible="#{cursoBean.estaCadastrando()}">
                        <p:focus context="painelCadastro" />
                        <h:form>
                              <br />
                              <h:panelGrid columns="2" cellpadding="10">
                                    <p:outputLabel value="Nome:" for="edtNomeCurso" />
                                    <p:inputText id="edtNomeCurso"
value="#{cursoBean.curso.nome}"
                                          maxlength="60" size="100" required="true"
                                          requiredMessage="Informe o nome do
curso..." />

                                    <p:outputLabel value="Descrição:"
for="edtDescCurso" />
                                    <p:inputText id="edtDescCurso"
value="#{cursoBean.curso.descricao}"
                                          maxlength="100" size="100" />
                              </h:panelGrid>
                              <br />
                              <br />
                              <p:separator />
                              <h:panelGrid columns="2">
                                    <p:commandButton value="Confirmar" icon="ui-icon-
check"
                                          action="#{cursoBean.gravar()}"
process="@form"
                                          update=":painelCadastro :painelLista
:tabelaPrincipal" />
                                    <p:commandButton value="Cancelar" icon="ui-icon-
close"
                                          action="#{cursoBean.limpar()}"
process="@this"
                                          update=":painelCadastro :painelLista" />
                              </h:panelGrid>
                        </h:form>
                  </p:panel>
            </ui:define>
</ui:composition>
```

Listagem 3 – curso.xhtml

Pode-se ver que a página de cursos utiliza dois painéis principais. Um é responsável por mostrar a tabela que lista os cursos já cadastrados, permitindo paginar a exibição dos dados e aumentar o número de linhas a serem exibidas sem a necessidade de utilizar Java Script, apenas utilizando o componente “dataTable” do Primefaces. O outro painel contém o formulário com os campos necessários para cadastrar um curso. Foi utilizado o atributo “visible” desses painéis, chamando uma função de um ManagedBean que retorna um valor

booleano. Apenas um desses painéis ficará visível para o usuário. Isso é controlado pelo ManagedBean, que possui uma variável chamada “operacao”. Essa variável é do tipo “String” e pode ter três valores: “inserindo”, “alterando” e “listando”. Se a operação estiver como “listando” apenas o painel que lista os cursos estará visível. Caso contrário, apenas o painel com o formulário estará visível. A Listagem 4 mostra o ManagedBean completo, com os métodos que alteram a variável “operacao”.

```
package br.com.bean;

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

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;

import br.com.dao.DAO;
import br.com.modelo.Curso;
import br.com.util.JSFUtil;

@ManagedBean
@ViewScoped
public class CursoBean implements Serializable {

	private static final long serialVersionUID = 1L;

	private Curso curso;
	private String operacao;
	private String tituloPainelCadastro;

	public CursoBean() {
		limpar();
	}

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

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

	public String getOperacao() {
		return operacao;
	}

	public void setOperacao(String operacao) {
		this.operacao = operacao;
	}

	public String getTituloPainelCadastro() {
		return tituloPainelCadastro;
	}

	public void setTituloPainelCadastro(String tituloPainelCadastro) {
		this.tituloPainelCadastro = tituloPainelCadastro;
	}
```

```
	public List<Curso> getCursos() {
		if (!estaCadastrando()) {
			return new DAO<Curso>(Curso.class).listaTodos();
		} else {
			return null;
		}
	}

	public void prepararIncluir(){
		setCurso(new Curso());
		setOperacao("incluindo");
		setTituloPainelCadastro("Incluir Curso");
	}

	public void prepararAtualizar(Long idCurso){
		Curso curso = new DAO<Curso>(Curso.class).buscaPorId(idCurso);

		if (curso != null){
			setCurso(curso);
			setOperacao("alterando");
			setTituloPainelCadastro("Alterar Curso");
		}
	}

	public boolean gravar() {
		if (getOperacao().equals("incluindo")) {
			new DAO<Curso>(Curso.class).adiciona(getCurso());
		} else if (getOperacao().equals("alterando")) {
			new DAO<Curso>(Curso.class).atualiza(getCurso());
		}

		JSFUtil.adicionaMensagemInfo("Operação realizada com sucesso!");

		limpar();

		return true;
	}

	public boolean remover(Long idCurso) {
		Curso curso = new DAO<Curso>(Curso.class).buscaPorId(idCurso);

		if (curso == null) {
			return false;
		}

		new DAO<Curso>(Curso.class).remove(curso);
		JSFUtil.adicionaMensagemInfo("Curso removido com sucesso!");
		return true;
	}

	public void limpar() {
		setCurso(null);
		setOperacao("listando");
	}

	public Boolean estaCadastrando() {
		return 		getOperacao().equals("incluindo") 		||
getOperacao().equals("alterando");
```

```
	}

}
```

Listagem 4 – CursoBean.java

Pode-se ver que o ManagedBean “CursoBean.java” possui a anotação “ViewScoped”. Dessa forma, essa classe ficará criada enquanto a página que faz referência à mesma estiver aberta no navegador. Essa classe também possui um atributo do tipo “Curso” que é o nosso modelo. É através do ManagedBean que a camada de visão se comunica com a camada de modelo. Essa classe ainda utiliza outra classe, “JSFUtil.java”, que foi criada apenas para a adição de Mensagens para o usuário. O código da classe “JSFUtil.java” está visível na Listagem 5.

```
package br.com.util;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

public class JSFUtil {

	public static void adicionaMensagemErro(String mensagem) {
		FacesContext.getCurrentInstance().addMessage(":messages",
			new	FacesMessage(FacesMessage.SEVERITY_ERROR,	mensagem,
mensagem));
	}

	public static void adicionaMensagemInfo(String mensagem) {
		FacesContext.getCurrentInstance().addMessage(":messages",
				new	FacesMessage(FacesMessage.SEVERITY_INFO,	mensagem,
mensagem));
	}

}
```

Listagem 5 – JSFUtil.java

A classe “JSFUtil.java” (Listagem 6) contém dois métodos: um para adicionar mensagens de erro e outro para adicionar mensagens de informação. A mensagem é passada por parâmetro nos dois casos. O primeiro atributo do método “addMessage” é o “id” de um componente do PrimeFaces. Nesse caso é o “id” do componente “growl” definido na página modelo. Esse componente possui o atributo “autoupdate” e foi definido o valor “true” para o mesmo, dessa forma, todas as mensagens adicionadas ao contexto atual serão automaticamente renderizadas por esse componente por meio de Ajax.

```
package br.com.dao;

import java.util.List;
import java.util.Map;
```

```
import br.com.bean.generic.EstadoView;

public interface IDAO<T> {

	void adicionaOuAtualiza(T t, EstadoView estado);
	void remove(T t);
	List<T> listaTodos();
	T buscaPorId(Long id);
	T buscaUnicoPorNamedQuery(String namedQuery, Map<String, Object>
parametros);
	List<T> buscaVariosPorNamedQuery(String namedQuery, Map<String, Object>
parametros);
	Long buscaLongPorNamedQuery(String namedQuery, Map<String, Object>
parametros);

}
```

Listagem 6 – IDAO.java

Na Listagem 6 é apresentado o código da Interface IDAO.java, implementada pela classe DAO.java, apresentada na Listagem 7.

A classe DAO.java (Listagem 7) possui todos os métodos responsáveis por acessar o banco de dados e realizar operações ou consultas. Ela é uma classe genérica, não havendo a necessidade de criar uma classe DAO para cada entidade, como é feito normalmente, basta passar o tipo da classe no construtor. Para as consultas mais complexas, existem dois métodos: um é o "buscaUnicoPorNamedQuery" que retorna apenas um valor e o outro é o "buscaVariosPorNamedQuery" que retorna uma lista de valores. Ambos aceitam os mesmos parâmetros: o nome da consulta e os parâmetros utilizados na consulta. Existe, ainda, um método chamado "buscaLongPorNamedQuery" que também aceita os mesmos parâmetros, mas deve ser utilizado quando se deseja fazer uma consulta para retornar uma contagem. Essa contagem retorna um valor do tipo java.lang.Long.

```
	package br.com.dao;

	import java.util.List;
	import java.util.Map;
	import java.util.Map.Entry;

	import javax.persistence.NoResultException;
	import javax.persistence.Query;
	import javax.persistence.EntityManager;
	import javax.persistence.criteria.CriteriaQuery;

	import br.com.bean.generic.EstadoView;
	import br.com.util.JPAUtil;

	public class DAO<T> implements IDAO<T> {

		private final Class<T> classe;

		public DAO(Class<T> classe) {
```

```
                this.classe = classe;
        }

        @Override
        public void adicionaOuAtualiza(T t, EstadoView estado) {
                EntityManager em = new JPAUtil().getEntityManager();
                em.getTransaction().begin();

                if (estado == EstadoView.INSERINDO) {
                        em.persist(t);
                } else {
                        em.merge(t);
                }

                em.getTransaction().commit();
                em.close();
        }

        @Override
        public void remove(T t) {
                EntityManager em = new JPAUtil().getEntityManager();
                em.getTransaction().begin();

                em.remove(em.merge(t));

                em.getTransaction().commit();
                em.close();
        }

        @Override
        public List<T> listaTodos() {
                EntityManager em = new JPAUtil().getEntityManager();

                CriteriaQuery<T> query =
em.getCriteriaBuilder().createQuery(classe);
                query.select(query.from(classe));

                List<T> lista = em.createQuery(query).getResultList();

                em.close();
                return lista;
        }

        @Override
        public T buscaPorId(Long id) {
                EntityManager em = new JPAUtil().getEntityManager();
                T instancia = em.find(classe, id);
                em.close();
                return instancia;
        }

        @SuppressWarnings("unchecked")
        @Override
        public T buscaUnicoPorNamedQuery(String namedQuery, Map<String,
Object> parametros) {
                T result = null;

          try {
```

```
                    result = (T) criarQuery(namedQuery,
parametros).getSingleResult();
                } catch (NoResultException nre) {
                    System.out.println("NoResultException: " + nre.getMessage());
                } catch (Exception e) {
                    System.out.println("Erro ao rodar a query: " + e.getMessage());
                    e.printStackTrace();
                }

                return result;
            }

            @SuppressWarnings("unchecked")
            @Override
            public List<T> buscaVariosPorNamedQuery(String namedQuery,
Map<String, Object> parametros) {
                    List<T> result = null;

                try {
                    result = (List<T>) criarQuery(namedQuery,
parametros).getResultList();
                } catch (NoResultException nre) {
                    System.out.println("NoResultException: " + nre.getMessage());
                } catch (Exception e) {
                    System.out.println("Erro ao rodar a query: " + e.getMessage());
                    e.printStackTrace();
                }

                return result;
            }

            @Override
            public Long buscaLongPorNamedQuery(String namedQuery, Map<String,
Object> parametros) {
                    Long result = null;

                    try {
                            result = (Long) criarQuery(namedQuery,
parametros).getSingleResult();
                    } catch (NoResultException nre) {
                            System.out.println("NoResultException: " +
nre.getMessage());
                    } catch (Exception e) {
                            System.out.println("Erro ao rodar a query: " +
e.getMessage());
                            e.printStackTrace();
                    }

                    return result;
            }

            private Query criarQuery(String namedQuery, Map<String, Object>
parametros) {
                    EntityManager em = new JPAUtil().getEntityManager();
                Query query = em.createNamedQuery(namedQuery);

                if (parametros != null && !parametros.isEmpty()) {
                    popularQueryComParametros(query, parametros);
                }
```

```
            return query;
        }

        private void popularQueryComParametros(Query query, Map<String,
Object> parameters) {

            for (Entry<String, Object> entry : parameters.entrySet()) {
                query.setParameter(entry.getKey(), entry.getValue());
            }
        }

    }
```

Listagem 7 – DAO.java

A classe LoginBean.java (Listagem 8) é um ManagedBean que usa o escopo de sessão, ou seja, enquanto a sessão estiver ativa ele ficará instanciado. Essa classe é utilizada para validar o acesso ao sistema, procedimento que é feito pelo método "autenticarUsuario", mas também possui um método chamado "cadastrarUsuarioPadraoSeNecessario" que é chamado no construtor. Esse método verifica a existência do usuário "admin" no banco de dados, caso ele não exista o sistema o cadastra, informando a senha "admin". Na classe LoginBean também é feita a alteração dos dados do usuário logado.

```
package br.com.bean;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

import javax.annotation.PreDestroy;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

import org.primefaces.context.RequestContext;

import br.com.bean.generic.EstadoView;
import br.com.dao.DAO;
import br.com.modelo.Usuario;
import br.com.util.JSFUtil;

@ManagedBean
@SessionScoped
public class LoginBean implements Serializable {

    private static final long serialVersionUID = 1L;

    public static final long ADMINISTRADOR = 2;
    public static final long RESPONSAVEL = 1;
    public static final long AVALIADOR = 0;

    private Usuario usuario;
    private String login;
```

```
    private String senha;
    private Boolean alterandoSenha;

    public LoginBean() {
       cadastrarUsuarioPadraoSeNecessario();
    }

    public String getLogin() {
        return login;
    }

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

    public String getSenha() {
        return senha;
    }

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

      public Usuario getUsuario() {
            return usuario;
      }

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

      public Boolean getAlterandoSenha() {
            return alterandoSenha;
      }

      public void setAlterandoSenha(Boolean alterandoSenha) {
            this.alterandoSenha = alterandoSenha;
      }

      public void cadastrarUsuarioPadraoSeNecessario() {
            Map<String, Object> parametros = new HashMap<String, Object>();
            parametros.put("pLogin", "admin");

            Usuario usuario = new
DAO<Usuario>(Usuario.class).buscaUnicoPorNamedQuery(
                        Usuario.BUSCA_POR_LOGIN, parametros);

            if (usuario == null) { // se não existe o usuario padrao, cadastramos
                  usuario = new Usuario();
                  usuario.setNome("Administrador");
                  usuario.setLogin("admin");
                  usuario.setSenha("admin");
                  usuario.setPerfil(2); //administrador
                  new DAO<Usuario>(Usuario.class).adicionaOuAtualiza(usuario,
EstadoView.INSERINDO);
            }

      }
```

```
	public String autenticarUsuario() {
	if (getLogin() != null && getSenha() != null) {

		Map<String, Object> parametros = new HashMap<String, Object>();
		parametros.put("pLogin", getLogin());
		parametros.put("pSenha", getSenha());

		setUsuario(new DAO<Usuario>(Usuario.class).buscaUnicoPorNamedQuery(
				Usuario.BUSCA_POR_LOGIN_SENHA, parametros));

		if (getUsuario() != null) {

	FacesContext.getCurrentInstance().getExternalContext().getSessionMap().put("
usuario", getUsuario());
			return "/admin/inicial?faces-redirect=true";
		}
	}

	  JSFUtil.adicionaMensagemErro("Login e/ou senha inválidos...");
	  return "";
    }

	public void prepararAlterarDadosUsuarioLogado() {
		setAlterandoSenha(false);
	}

	public void prepararAlterarSenhaUsuarioLogado() {
		setAlterandoSenha(true);
	}

	public void alterarDadosUsuarioLogado() {
		try {
			new
DAO<Usuario>(Usuario.class).adicionaOuAtualiza(getUsuario(),
EstadoView.ALTERANDO);
			RequestContext context = RequestContext.getCurrentInstance();
			if (getAlterandoSenha()) {
				context.execute("PF('dlgSenha').hide();");
			} else {
				context.execute("PF('dlgUsuario').hide();");
			}
			JSFUtil.adicionaMensagemInfo("Operação realizada com
sucesso...");
		} catch (Exception e) {
			e.printStackTrace();
			JSFUtil.adicionaMensagemErro("Não foi possível realizar a
operação...");
		}
	}

	public String logoff() {
		setUsuario(null);
		setLogin("");
	  setSenha("");

FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
	  return "/acesso/login?faces-redirect=true";
	}
```

```
	public boolean perfilAvaliador() {
		return getUsuario().getPerfil() == AVALIADOR;
	}

	public boolean perfilResponsavel() {
		return getUsuario().getPerfil() == RESPONSAVEL;
	}

	public boolean perfilAdministrador() {
		return getUsuario().getPerfil() == ADMINISTRADOR;
	}

	public boolean usuarioPadrao() {
		if (getUsuario() != null) {
			return getUsuario().getLogin().equals("admin");
		} else {
			return false;
		}
	}

	@PreDestroy
	public void destroy() {
		logoff();
	}

}
```

Listagem 8 – LoginBean.java

5 CONCLUSÃO

O objetivo deste trabalho foi implementar um sistema para registrar a avaliação de trabalhos acadêmicos apresentados perante bancas. A aplicação foi desenvolvida para a plataforma *web*, pela facilidade de acesso e manutenção, utilizando recursos que a caracterizam como uma RIA. Para esse desenvolvimento, foram utilizadas diversas ferramentas e tecnologias.

Uma destas ferramentas é o Hibernate, um *framework* ORM, que tem como principal finalidade mapear o "mundo orientado a objetos" para o "mundo relacional" automaticamente de forma transparente para o desenvolvedor. Dessa forma, não é necessária a utilização de linguagem SQL dentro do código.

Outra ferramenta é o Primefaces, biblioteca que implementa a especificação do JSF. O JSF busca unir as principais vantagens entre aplicações *desktop* e aplicações *web*, utilizando componentes ricos, sem a necessidade da utilização de JavaScript e HTML. O Primefaces, como outras bibliotecas semelhantes, tem a finalidade de melhorar os componentes já existentes na especificação e, ao mesmo tempo, incluir novos.

O Primefaces 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 desvantagem, que pode preocupar desenvolvedores de aplicações *desktop*, é o uso de eventos, que os componentes realizam utilizando Ajax. Utilizar dois ou mais eventos Ajax do mesmo componente pode resultar em uma ação inesperada. Esse é um problema que ainda necessita de uma solução adequada.

Como trabalhos futuros ressaltam-se a implementação de algumas funcionalidades como o envio de *email* para a banca e aviso de avaliações pendentes, bem como a geração automática da ata.

REFERÊNCIAS

BLAHA, Michael; JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**. 2ª ed. Rio de Janeiro: Elsevier, 2006.

BRAMBILLA, Marco; PRECIADO, Juan Carlos; LINAJE, Marino; SANCHEZ-FIGUEROA, Fernando. **Business process-based conceptual design of rich internet applications**. In: 8th International Conference on Web Engineering (ICWE 08), IEEE CS Press, 2008, p. 155-161.

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.

LAWTON, George. New ways to build rich Internet applications. **Computer. Published by the IEEE Computer Society**, p. 10-12, 2008.

LINAJE, Marino; PRECIADO, Juan Carlos; SÁNCHEZ-FIGUEROA, Fernando. A method for model based design of rich internet application interactive user interfaces. **Lecture Notes in Computer Science**, v. 4607, p. 226-241, 2007.

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.

PRECIADO, Juan Carlos; LINAJE, Marino; COMAI, Sara; SANCHEZ-FIGUEROA, Fernando. **Designing rich internet applications with web engineering methodologies**. In: 9th IEEE Workshop Web Site Evolution (WSE 07), IEEE CS Press, 2007, p. 23–30.

PRESSMAN, Roger. **Engenharia de software**, 5ª ed. 2002. Rio de Janeiro: McGrawHill.

ROSSI, Gustavo, URBIETA; Matias, GINZBURG; Jeronimo, DISTANTE; Damiano, GARRIDO, Alejandra. **Refactoring to rich internet applications. A model-driven approach**. In: Eighth International Conference on Web Engineering, 2008, p. 1-12.