UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DIOGO SATDL

SISTEMA PARA GERENCIAMENTO DE EVENTOS EM JAVA EE

PATO BRANCO
2015

DIOGO SATDL

SISTEMA PARA GERENCIAMENTO DE EVENTOS EM JAVA EE

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, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO

2015

ATA Nº: **271**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **DIOGO SATDL**.

Às 14:20 hrs do dia 24 de novembro de 2015, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Vinicius Pegorini (Orientador), Beatriz Terezinha Borsoi (Convidada) e Pablo Gauterio Cavalcanti (Convidado), para avaliar o Trabalho de Diplomação do aluno Diogo Satdl, matrícula 981052, sob o título **Sistema de gerenciamento de eventos em javaee**; 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 15:10 hrs foi encerrada a sessão.

Prof. Vinicius Pegorini, M.Sc.
Orientador

Profa. Beatriz Terezinha Borsoi, Dr.
Convidada

Prof. Pablo Gauterio Cavalcanti, Ph.D
Convidado

Profa. Soelaine Rodrigues Ascari, M. Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

SATDL, Diogo. Sistema para gerenciamento de eventos em Java EE. 2015. 79f. - Trabalho de Conclusão de Curso. Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

Comissões responsáveis pela organização de eventos geralmente encontram uma série de dificuldades na ordenação e realização de suas atividades, diversas vezes contando apenas com um controle manual do processo. Tal realidade é normalmente encontrada em instituições de ensino, as quais ao longo do período letivo desenvolvem uma série de eventos que exigem agilidade e segurança em seu controle. Visando melhorias tanto para a comissão organizadora quanto para os participantes, permitindo um gerenciamento completo de todo o processo, foi desenvolvido esse projeto. Neste trabalho é reportada a análise e desenvolvimento de um sistema de gerenciamento de eventos para instituições de ensino com uso da linguagem de programação Java. Para armazenamento de dados foi utilizado o banco de dados relacional MySQL. Também foram utilizadas a ferramenta de modelagem *Visual Paradigm*, e a ferramenta administrativa de banco de dados MySQL *WorkBench*. Como resultado foi gerado um sistema capaz de gerenciar e acompanhar as atividades desenvolvidas pela instituição.

Palavras-chave: Linguagem de Programação Java. *MySQL*. *Visual Paradigm*. Sistema para Gerenciamento de Eventos.

ABSTRACT

SATDL, Diogo. System for event management in JavaEE. 2015. 79f - Trabalho de Conclusão de Curso. Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2015.

Committees responsible for organizing events generally pass through several difficulties in their activities when they have only a manual process control. This reality is usually found in educational institutions, which over the semester develop several events that require agility and security in their control. In order of improve the tasks to the organizing committee and for the participants, it was developed this project, which allow the complete management of an event. This study reports the analysis and development of an event management system for educational institutions, for the development is used the Java programming language. For data storage was used the relational database MySQL. We used the Visual Paradigm as modeling tool, and the administrative tool for MySQL database called WorkBench. As a result was generated a system to manage and monitor the activities of the institution.

Keywords: Java Programming Language. *MySQL*. *Visual Paradigm*. *Event Management System*.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS E ACRÔNIMOS

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
ARPA	*Advanced Research Projects Agency*
CASE	*Computer Aided Software Engineering*
CDI	*Context Dependency Injection*
CERN	*Conseil Européen pour la Recherche Nucléaire*
CPF	Cadastro de Pessoa Física
CRUD	*Create, Read, Update and Delete*
CSS	*Cascading Style Sheets*
DAO	*Data Object Access*
DER	Diagramas de Entidades e Relacionamentos
DOM	*Document Object Model*
EJB	*Enterprise JavaBeans*
GUI	Graphical User Interface
HQL	*Hibernate Query Language*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JAX-RS	*Java API for RESTful Web Services*
JAX-WS	*Java API for XML-Based Web Services*
JEE	*Java Enterprise Edition*
JMS	*Java Message Service*
JPA	*Java Persistence API*
JSF	*Java Server Faces*
JSP	*Java Server Pages*
JTA	*Java Transaction API*
LAN	*Local Area Network*
MVC	*Model View Controller*
ORM	*Object-Relational Mapping*
REST	*Representational State Transfer*
RF	Requisito Funcional
RMI	*Remote Method Invocation*
RNF	Requisito Não Funcional.
SOAP	*Simple Object Access Protocol*
SQL	*Structure Query Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
XHTML	*eXtensible Hypertext Markup Language*
XML	*eXtensible Markup Language*
XSTL	*EXtensible Stylesheet Language Transformations*

SUMÁRIO

1. INTRODUÇÃO

Este capítulo apresenta as considerações inicias, com uma visão geral do trabalho, os objetivos e a justificativa do mesmo e a organização do texto.

1.1 CONSIDERAÇÕES INICIAIS

As instituições de ensino oferecem ao decorrer do ano letivo uma série de eventos de caráter institucional e/ou educacional. Por evento educacional entendem-se semanas acadêmicas, treinamentos, cursos e palestras. Eventos institucionais por sua vez, correspondem a reuniões de trabalho, mutirões e ações de cidadania.

É comum que o gerenciamento de tais eventos seja realizado utilizando-se de fichas de inscrição, listas de presença e emissão de certificados feitos manualmente. Tal método torna-se, na maioria das vezes, de difícil controle por parte da instituição e da comissão organizadora do evento, além de ser trabalhoso, coletar assinaturas em listas de presença.

O controle desses eventos exige agilidade e segurança das informações coletadas, pois é necessário o controle de presença e a emissão de certificados para todos os participantes. Para isso, necessita-se de um sistema capaz de armazenar, processar e gerar respostas padronizadas.

Visando possibilitar uma alternativa aos controles diversos e facilitar seu gerenciamento, propõe-se neste trabalho, automatizar o processo de gestão de eventos, por meio de uma aplicação que gerencie cada evento desde seu início até a finalização do mesmo. Beneficiando tanto a comissão organizadora quanto os participantes de cada atividade de um evento.

Para facilitar o acesso, o aplicativo desenvolvido é para ambiente Internet e permite a identificação do participante por leitura de código de barras, visando, assim, agilizar a entrada no evento e facilitar a inscrição e recebimento de certificado.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um sistema *Web* para realizar o controle dos principais processos envolvidos na organização e na realização de eventos.

1.2.2 Objetivos Específicos

- Implementar um controle de acesso dos usuários.
- Possibilitar o cadastro e manutenção completa de eventos e atividades.
- Controlar inscrições em eventos, listas de presença e controle de entradas com crachá ou número do registro acadêmico, via leitor de código de barras.
- Realizar o envio de certificados via e-mail.

1.3 JUSTIFICATIVA

Justifica-se o desenvolvimento de um sistema *web* para o gerenciamento dos principais processos da organização de eventos, para possibilitar que a comissão organizadora efetue um controle completo dos usuários inscritos e sua presença em cada palestra, minicurso e demais atividades. Por exemplo, o controle das presenças dos participantes das atividades de um evento, permite aos organizadores o envio automático dos certificados após a realização da atividade sem a necessidade de consultas manuais à lista de presença.

O controle do processo de cadastro de uma atividade permitirá associar a atividade a um local especifico, controlando o número máximo de inscritos, horário de início e de fim. Essas informações, além de auxiliar a comissão organizadora, também permitem que os participantes dos eventos possam realizar a inscrição de maneira mais fácil, assim como a consulta de vagas disponíveis em cada evento.

Sendo assim, a proposta de construção de um sistema *web* capaz de automatizar o processo de controle de um evento, contribui com a comissão organizadora na eficácia do controle de inscrições nas atividades, no monitoramento

de presenças e na emissão de certificados. O sistema também auxiliará aos usuários do evento com informações relacionadas ao local e os horários das atividades, além de permitir a emissão do próprio certificado.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em capítulos, os quais apresentam no decorrer do seu desenvolvimento teórico e prático o surgimento de um sistema *web* para a realização e controle dos principais processos envolvidos na organização e realização de eventos.

No Capítulo 1 são apresentadas as considerações iniciais do trabalho, apresentando a ideia e justificativa do sistema, expondo os objetivos propostos pelo acadêmico, os quais servem como base para todo o desenvolvimento da ideia que até então foi proposta.

O Capítulo 2 apresenta o referencial teórico, o qual fundamenta a proposta para o sistema desenvolvido, apresentando uma visão sobre a evolução do desenvolvimento *web* e seus principais conceitos.

Já no Capítulo 3 é possível visualizar os materiais e o método utilizados no desenvolvimento do presente trabalho, assim como todas as ferramentas e tecnologias necessárias para o desdobramento do sistema.

No Capítulo 4 são expostos os resultados, contendo a apresentação do sistema, modelagem por meio de diagramas de entidade e relacionamento. A implementação é apresentada por meio de telas e exemplos de códigos.

Por fim, o Capítulo 5 contém as considerações finais do acadêmico, apresentando os resultados decorrentes de todo o desenvolvimento do trabalho, bem como as conclusões associadas à importância de um sistema *web* para o gerenciamento de atividades.

2. REFERENCIAL TEÓRICO

2.1 Modelo Cliente-Servidor

Para Tanenbaum (2006) um sistema distribuído é uma coleção de computadores independentes que se apresentam aos seus usuários como um sistema único e coerente.

Os primeiros computadores, conhecidos como mainframes eram equipamentos de grande porte que executavam instruções por intermédio de fitas de papel gravadas ou codificadas. Os mesmos ficavam conectados aos terminais sem capacidade de processamento que faziam interação por linha de comando.

Segundo Tanenbaum (2006), entre 1945 e 1985, computadores eram grandes e caros e até minicomputadores chegavam a custar dezenas de dólares. Devido a isso, apenas algumas organizações possuíam computadores, que trabalhavam isolados uns dos outros, devido ao fato de não existir um meio para conectá-los. Então, a partir de meados dos anos 80, avanços sem precedentes na tecnologia, permitiram o desenvolvimento de poderosos microprocessadores. Máquinas custando centenas de dólares e executando apenas uma instrução por segundo deram lugar a máquinas que executavam um milhão de instruções por segundo, e custavam uma fração do preço anterior.

Ainda de acordo com Tanenbaum (2006), um segundo avanço tecnológico merece destaque, a invenção das redes de computadores. As *Local Area Network* (LAN), permitiram que centenas de computadores fossem conectados transferindo informações a taxas de até 10 bilhões de bits por segundo. Os resultados desses avanços tecnológicos resumem o surgimento das redes de computadores e os sistemas distribuídos, em contraste com os sistemas antes utilizados que consistiam de apenas um computador e seus periféricos.

Tais avanços permitiram o surgimento do computador pessoal e também da interface de usuário, com aplicativos que poderiam ser instalados e operados via cliques em botões, o que se tornou um padrão até hoje. Com a instalação de aplicativos surgiram então problemas relacionados a espaço de armazenamento, capacidade de processamento e manutenção desses softwares. Um computador, uma vez conectado a uma rede, pode executar programas localmente ou de forma compartilhada, usando recursos como dados, capacidade de processamento e

armazenamento de outras máquinas ou de um servidor. Esse modelo de arquitetura é conhecido como cliente/servidor.

Neste modelo, os dados são armazenados em computadores chamados servidores, mantidos e hospedados por administradores de sistemas. Máquinas mais simples chamadas clientes, fazem acesso remoto a tais dados através de conexão a uma mesma rede. Tal modelo é aplicável quando ambas as máquinas se encontram no mesmo espaço, ou muito distantes como quando por exemplo uma página de internet é acessada (TANENBAUM, STEEN; 2006).

2.2 A World Wide Web

A Internet foi concebida em 1969 pela Agência de Pesquisa e Projetos Avançada dos Estados Unidos (ARPA), ficando conhecida como ARPANET. O objetivo inicial da ARPANET era que computadores de pesquisa de uma universidade pudessem “conversar” com computadores de outras universidades.

Segundo Kurose (2011), até o final da década de 1970, cerca de duas centenas de *hosts* foram conectados a ARPANET. Até o final da década de 1980 o número de *hosts* conectados à internet pública, uma confederação de redes muito parecida com a internet de hoje, iria chegar a centenas de milhares. A década de 1980 seria um momento de enorme crescimento.

Durante a década de 90 acontece o crescimento do *World Wide Web*, que trouxe a Internet para dentro das casas e empresas. A *Web* foi inventada na Organização Europeia para a Pesquisa Nuclear (CERN) por Tim Berners-Lee entre 1989 e 1991. Berners-Lee e seus colegas desenvolveram versões iniciais do *HyperText Markup Language* (HTML), *Hypertext Transfer Protocol* (HTTP), um servidor *Web* e um navegador, sendo estes os quatro componentes chaves da *Web*.

Para Coulouris *et al.* (2013), uma característica importante da *web*, é que ela fornece uma estrutura de hipertexto entre os documentos que armazena, refletindo a necessidade dos usuários de organizar seus conhecimentos. Isso significa que os documentos contêm links (ou *hyperlinks*) - referências para outros documentos e recursos que também estão armazenados na *web*.

Conforme aponta Colourois *et al.* (2013), a *Web* tem evoluído sem mudar sua arquitetura básica, que é baseada em três componentes tecnológicos padrão principais:

- HTML, que é uma linguagem para especificar o conteúdo e o *layout* de páginas de forma que elas possam ser exibidas pelos navegadores Web.
- *Uniform Resource Locator (*URL), que identificam os documentos e outros cursos armazenados como parte da web.
- Uma arquitetura de cliente/servidor, com regras padrão para interação (o protocolo HTTP), por meio das quais os navegadores e outros clientes buscam documentos e outros recursos dos servidores. Na Figura 1 estão ilustrados os passos de uma requisição HTTP.

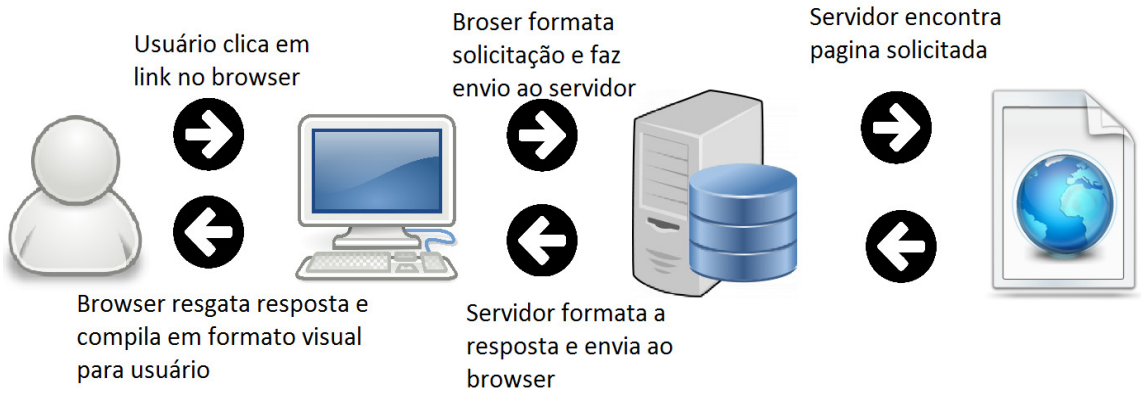

Figura 1 - Requisição - Resposta HTTP

De acordo com Mikkonen e Taivalsaari (2008), atualmente os quatro componentes fundamentais da Web são: HTML, *Cascading Style Sheets (*CSS), linguagem de script como *JavaScript* e *Document Object Model (*DOM). Sendo:

- HTML: é um sistema de *tags* de códigos que definem elementos na página, os quais criam a estrutura de um documento para uso na *web* (TEAGUE, 2010).
- CSS: é uma linguagem usada para especificar a aparência visual de uma página *web* (TEAGUE, p.34, 2010).
- *JavaScript*: é uma linguagem orientada a objetos interpretada, usada para escrever aplicações do lado cliente, o que significa que seu código é enviado ao usuário quando a página é carregada. O código é então

executado, por um interpretador de Java script, incluído como parte do browser cliente (BROOKS, 2007).

- DOM: é uma interface de programação de aplicativo *Application Programming Interface* (API)) para HTML válido e documentos XML bem formados. Ela define a estrutura lógica de documentos e a forma como um documento é acessado e manipulado. Com o *Document Object Model*, os programadores podem criar documentos, navegar na sua estrutura, e adicionar, modificar ou excluir elementos e conteúdo (W3C, 2010).

2.3 Web 2.0

Com a evolução da Internet e a demanda crescente por aplicações comerciais, o formato padrão baseado em HTML começou a apresentar limitações. Como resposta a essa demanda, surgiu a *Web* 2.0, termo que surgiu em uma conferência de *brainstorming* entre as empresas *O'Reilly Media* e a *MediaLive Internacional* no ano de 2004. Segundo O'Reilly (2005), pode-se visualizar o conceito de *Web* 2.0 como um conjunto de princípios e práticas que formam em conjunto um verdadeiro sistema solar de *Web sites* que demonstram alguns ou todos esses princípios.

De acordo com Murugesan (2007), a *Web* 2.0 não é apenas uma nova versão da mesma *Web*. Por exemplo:

- Facilita *design Web* flexível, reutilização criativa e atualizações.
- Fornece uma interface de usuário rica.
- Facilita a criação de conteúdo colaborativo e modificação.
- Permite a criação de novos aplicativos através da reutilização e combinação de diferentes aplicações na *Web* ou por combinação de dados e informações de diferentes fontes.
- Estabelece redes sociais das pessoas com interesses em comum.

Tais tendências se manifestam sob uma variedade de formas, nomes e tecnologias, do lado do cliente. Processamento *Asynchronous JavaScript and XML (*AJAX), *eXtensible Markup Language* (XML), blogs, *wikis*, *mashups* e conteúdo compartilhado, redes sociais e a participação do usuário. Aplicações *Web* 2.0 com comportamento semelhante a aplicações *desktop*, com maior velocidade e facilidade de uso podem ser facilmente observadas em aplicativos como *Youtube*,

GoogleDocs, *Office Online* e aplicações sociais, tais como o *Facebook*, o *Twitter*, e a enciclopédia *Wikipédia*.

Esses serviços *online* forçaram desenvolvedores a adotar novas tecnologias, linguagens e conceitos de construção de páginas *web*, tais como usabilidade, simplicidade de uso, interação com interface do site e navegabilidade.

Na Figura 2, é exibida uma visão geral do aplicativo *Word Online*, que faz parte do pacote *Office Online* oferecido gratuitamente pela *Microsoft*.

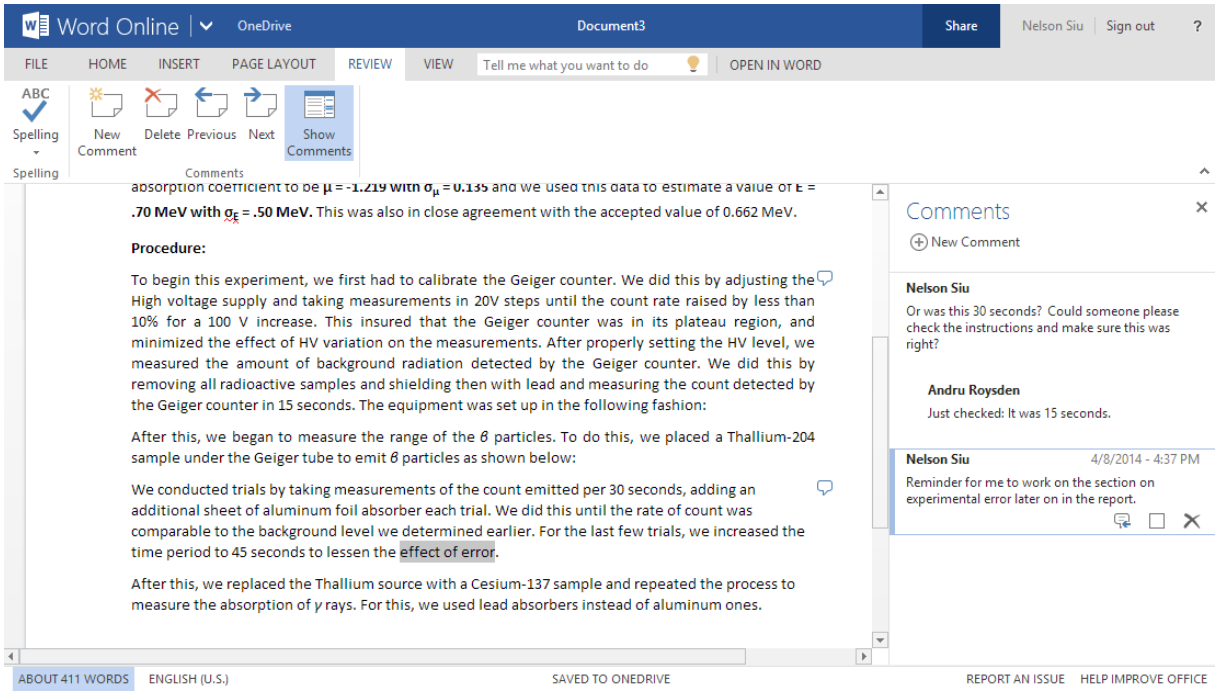

Figura 2 - Word Online - Serviço on-line. Parte do pacote Office Online.

Interfaces ricas, com a apresentada na Figura 2, auxiliam usuários a encontrar e manipular o conteúdo, bem como completar transações. O melhor exemplo de tecnologias utilizadas para o desenvolvimento de interfaces ricas é o AJAX.

2.4 AJAX

O termo AJAX foi inicialmente concebido por Jesse James Garret em seu artigo *"Ajax, A New Approach to Web Applications"*. De acordo com Garret (2005), AJAX não é uma tecnologia, e sim, várias, cada uma atuando de sua própria maneira, reunidas de uma forma poderosa. Desta forma, AJAX incorpora:

- Apresentação baseada em padrões usando *eXtensible Hypertext Markup Language* (XHTML) e CSS.
- Exposição dinâmica e interação com o DOM.
- Intercâmbio de dados e manipulação usando XML e *Extensible Stylesheet Language Transformations* (XSLT).
- Recuperação de dados assíncronos usando XMLHTTP *Request*.

A tecnologia AJAX permite que as páginas da *web* possam ser atualizadas de forma assíncrona por meio da troca de pequenas quantidades de dados com o servidor nos bastidores. Isto significa que é possível atualizar partes de uma página da *web* sem recarregar a página inteira (W3C, 2013).

A popularização da tecnologia começou em 2005 com o *Google Sugest*. Quando é iniciada a digitação na caixa de pesquisa do *Google*, por meio do uso de *JavaScript*, são enviadas ao servidor os caracteres informados e o servidor retorna uma lista de sugestões. Tal tecnologia cria uma interface dinâmica, visto que não é necessário atualizar toda a página novamente. Esse processo resume perfeitamente a proposta do AJAX, que é de atualizar pequenas partes da página dinamicamente sem a necessidade de atualizar novamente toda a página.

Segundo Zakas *at al*. (2007), muitos dos projetos do *Google*, como o *Suggest* e o *Maps*, envolvem uma única página que nunca é recarregada, mas atualizada constantemente. Tais inovações, que começaram a trazer muitas das acessibilidades de *softwares desktop*, para o interior do *browser*, foram aclamadas ao redor da *web* dando início a uma nova era no desenvolvimento *web*.

3. MATERIAIS E MÉTODO

Neste capítulo são apresentadas as ferramentas e as tecnologias utilizadas para a modelagem do sistema, a criação e a manutenção do banco de dados, e também, a linguagem de programação utilizada para implementar o software.

3.1 MATERIAIS

Foram necessárias para o desenvolvimento do sistema as ferramentas e tecnologias listadas a seguir:

- Java EE 7 - Linguagem para desenvolvimento do sistema.
- Java *Server Faces 2.2* - *Framework Model-View-Controller* (MVC) para construção de interfaces de usuário.
- *Prime faces* 5.2 - Biblioteca de componentes para o *JavaServer Faces* (JSF).
- MySQL 5.6 - Banco de Dados da aplicação.
- MySQL *Workbench* 6.3.5 - Para modelagem e gerenciamento do banco de dados.
- Spring *Framework* 4.1.6 - Para injeção de dependência e inversão de controle.
- *Spring Data* JPA 1.7.2 - Para controle de acesso a dados.
- *Spring Boot* 1.2.3 - Como automatização do ambiente e configuração do Spring**.**
- *Hibernate* 4.3.8 - *Framework Object-Relational Mapping* (ORM) utilizado pelo *Spring Data* para controle de dados.
- IntelliJ *Idea* 14.1 - Como *Integrated Development Environment* (IDE) de desenvolvimento Java.
- *JBoss WidFly* 9.0.1 - Como servidor de aplicações.
- *Visual Paradigm* 12.2 - Documentação da modelagem do sistema.

3.1.1 Java Enterprise Edition

Segundo Gonçalves (2014), o Java *Enterprise Edition* surgiu no final da década de 1990 e trouxe para a linguagem Java uma plataforma robusta para o desenvolvimento empresarial. Sendo uma linguagem orientada a objetos multi plataforma e possuindo uma vasta biblioteca de *frameworks* de desenvolvimento.

O *Java Enterprise Edition* (Java EE) (anteriormente chamado J2EE), é uma especificação guarda-chuva abraçando um conjunto padrão de tecnologias para o desenvolvimento do lado do servidor Java. Tecnologias Java EE incluem *Java Servlet*, *JavaServer Pages* (JSP), JSF, *Enterprise JavaBeans* (EJB), Contextos e Injeção de Dependência (CDI), *Java Messaging Service* (JMS), *Java Persistence API* (JPA), *Java* API para XML *Web Services* (JAX-WS), e API *Java* para *RESTful Web Services* (JAX -RS), entre outros (CMIL, MATLOKA, MARCHIONI, 2015).

Para Gonçalves (2014), o ambiente de execução Java EE define quatro tipos de componentes que uma aplicação deve apoiar:

- *Applets* são aplicações *Graphical User Interface* (GUI), interface gráfica de usuário, que são executados em um navegador *web*. Eles usam a API *Swing* para fornecer interfaces de usuário com um maior número de recursos.
- Os aplicativos são programas que são executados em um cliente. Eles são tipicamente *GUIs* ou programas *batchprocessing* que têm acesso a todas as instalações da camada intermediária Java EE.
- Aplicações *Web* (feitas de *servlets*, filtros de *servlet*, ouvintes de eventos *web*, páginas JSP e JSF), são executadas em um contêiner da *Web* e respondem às solicitações HTTP de clientes *web*. *Servlets* também suporte *Simple Object Access Protocol* (SOAP) e Representational State Transfer (REST) terminais de serviço *web*.
- As aplicações corporativas (feito de *Enterprise Java Beans, Java Message Service*, *Java Transaction API* (JTA), chamadas assíncronas, serviço de timer, RMI/IIOP), são executadas em um contêiner EJB. EJBs são componentes gerenciados por contêiner para o processamento de lógica de negócios transacionais. Eles podem ser acessados localmente e

remotamente por meio de *Remote Method Invocation* (RMI) ou HTTP para SOAP e serviços *Web RESTful* (GONÇALVES, 2014).

3.1.2 Java Server Faces

Para Bergsten (2004), o *Java Server Faces* simplifica o desenvolvimento de aplicações *web* e interfaces de usuário, definindo modelos de componentes de interface vinculados a processadores de requisições de ciclo de vida bem definidos. Isso permite:

- Programadores Java desenvolverem a aplicação no lado servidor sem se preocupar com detalhes HTTP e integrá-la à interface de usuário por meio de um modelo dirigido a eventos.
- Autores de páginas sem conhecimento de programação a trabalharem com os aspectos da interface por meio de componentes montados que encapsulam toda a lógica da integração com usuário, minimizando assim a necessidade de lógica de programação integrada diretamente nas páginas de interface.
- Fornecedores a desenvolver ferramentas poderosas para desenvolvimento tanto *frontend* quando *backend*.

Segundo Geary e Horstman (2012), o JSF contém todo o código necessário para a manipulação de eventos e a organização de componentes. Os programadores de aplicações podem ignorar esses detalhes e se dedicar apenas à lógica da aplicação.

O JSF não é o único *framework* com componentes baseados na *web*, mas é a camada de visão padrão do JEE. JSF está incluído em cada servidor de aplicações Java EE e pode ser facilmente adicionado a um contêiner web autônomo como o *Tomcat*.

3.1.3 Prime Faces

Segundo Caliskan e Varaksin (2015), o Prime Faces é uma biblioteca de componentes JSF leve com um arquivo ,jar, que não precisa de configuração e não

contém quaisquer dependências externas necessárias. Para começar com o desenvolvimento da biblioteca, é necessário apenas que o artefato esteja na biblioteca. O Prime Faces proporciona a seguinte lista de características, que fazem dele uma biblioteca de componentes fáceis de usar:

- Mais de 100 componentes de interface rica.
- Suporte a AJAX embutido.
- Não necessita de configurações.
- Não requer bibliotecas de terceiros para a maioria dos componentes.
- Integração com seletor de temas.
- Mais de 30 temas disponíveis.
- Suporte aos navegadores IE8+, Chrome, Firefox, Safari e Opera.

3.1.4 MYSQL

Ao longo dos anos, o termo banco de dados foi utilizado para descrever uma variedade de produtos, sistemas, dados e mecanismos de armazenamento e acesso a dados. Por exemplo, uma pequena empresa pode armazenar registros de folha de pagamento em arquivos individuais, enquanto uma empresa elétrica regional, utiliza um sistema integrado para manter registros de todos os seus clientes; gerar contas de energia elétrica para os clientes, e criar relatórios que definir padrões de uso de energia, demonstrações de lucros e perdas, ou mudanças na demografia do cliente. Em ambos os casos, as organizações podem se referir a cada um de seus sistemas como bancos de dados (SHELDON; MOES, 2005).

Para Sheldon e Moes (2005), tal como acontece com outros produtos de Sistema de Gerenciamento de Banco de Dados Relacional, em inglês Relational Database Management System (RDBMS), o MySQL fornece um rico conjunto de recursos que suportam um ambiente seguro para armazenar, manter e acessar a dados. MySQL é uma alternativa confiável, rápida e escalável para muitos dos RDBMSs comerciais disponíveis.

Segundo Dubois (2014), o sistema de gerenciamento de banco de dados MySQL é popular por muitas razões. É rápido e fácil de configurar, usar e administrar. Funciona sob muitas variedades de *Unix* e *Windows* e programas baseados no MySQL podem ser escrito em muitas linguagens de programação.

3.1.5 MySQL Workbench

O MySQL *Workbench* é uma ferramenta visual unificada para arquitetos de banco de dados, desenvolvedores e DBAs. O MySQL *Workbench* fornece modelagem de dados, desenvolvimento de SQL e ferramentas de administração abrangentes para configuração do servidor, administração de usuários, *backup* e muito mais. O MySQL *Workbench* está disponível para as plataformas *Windows*, *Linux e Mac OS X* (MYSQL, 2013).

Segundo a documentação oficial algumas das principais funcionalidades da ferramenta são:

- Projeto - MySQL Workbench permite que um DBA, desenvolvedor ou arquiteto de dados visualmente projete, modele, gere e gerencie bancos de dados.
- Desenvolver - Oferece ferramentas visuais para criar, executar e otimizar consultas SQL.
- Administrar - Fornece um console visual para administrar facilmente ambientes MySQL.
- Painel de desempenho - Fornece um conjunto de ferramentas para melhorar o desempenho de aplicações do MySQL.
- Migração de Banco de Dados - MySQL *Workbench* fornece uma solução completa e fácil de usar para migrar do *Microsoft* SQL *Server*, *Microsoft Access*, *Sybase* ASE, PostreSQL, e outras tabelas de RDBMS, objetos e dados para o MySQL (MySQL, 2013)

Na Figura 3, é apresentada a tela inicial da ferramenta destacando suas funcionalidades.

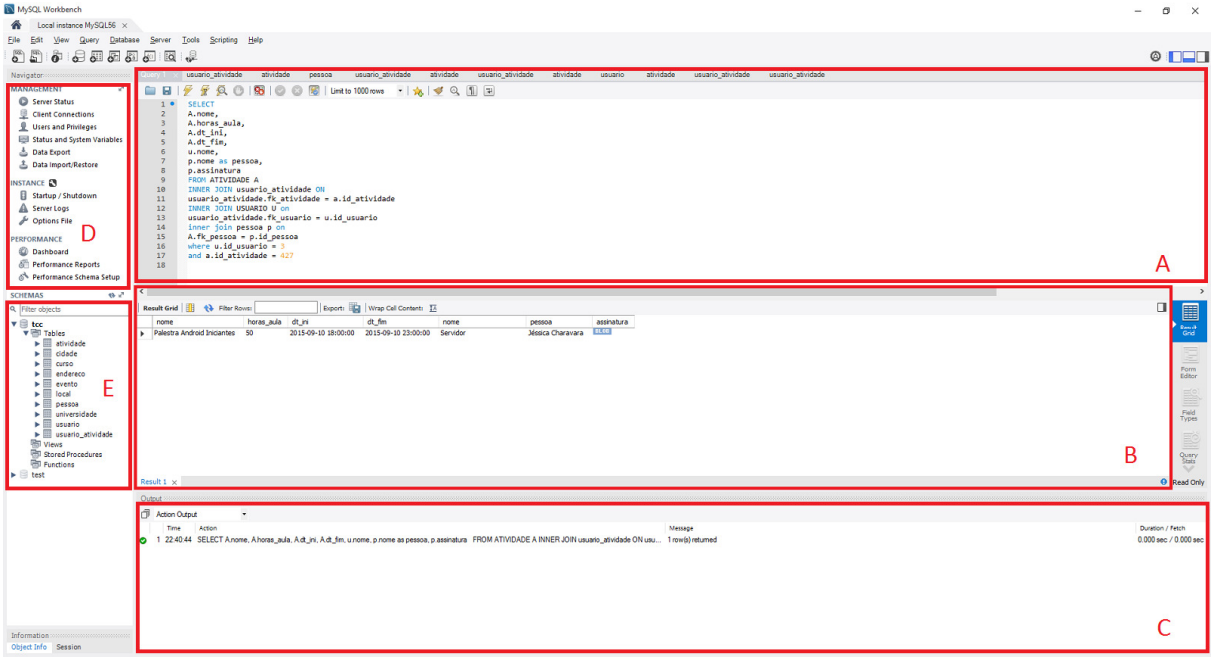

Figura 3 - Tela inicial do MySQL Workbench

Nas partes destacadas da Figura 3 é possível identificar:

a) Área de desenvolvimento - Local no qual são desenvolvidas e editadas as instruções SQL.
b) Grid de resultado - Registros resultantes da execução da instrução.
c) Resultado da ação - Local em que são apresentados os resultados da ação, como erros ou contagem de registros.
d) Gerenciamento do Servidor - Área que apresenta as principais funcionalidades do gerenciamento do servidor.
e) Bancos de Dados - Listagem dos bancos de dados do servidor conectado.

3.1.6 Spring Framework

O Spring Framework fornece um modelo abrangente de programação e configuração de aplicativos corporativos baseados em Java em qualquer tipo de plataforma de implementação. Sendo suas principais características (SPRING, 2013):

- Injeção de dependência.
- Programação Orientada a Aspectos incluindo gerenciamento de transações declarativas.

- Spring MVC aplicativo web e um quadro do serviço *Web RESTful*.
- Apoio fundamental para *Java Database Connectivity* (JDBC), JPA, JMS.

3.1.7 Spring Data JPA

Segundo a documentação oficial, o *Spring Data* JPA visa melhorar significativamente a camada de acesso a dados, reduzindo o esforço para a quantidade que é realmente necessário. O desenvolvedor escreve suas interfaces de repositório, incluindo métodos de busca personalizada e o *Spring* fornecerá à aplicação automaticamente. Assim, suas principais características são:

- Suporte sofisticado para construir repositórios com base no *Spring* e *JPA*.
- Suporte para predicados *Querydsl* e consultas JPA *type-safe*.
- Auditoria transparente da classe de domínio.
- Suporte de paginação, execução da consulta dinâmica, capacidade de integrar código de acesso de dados personalizado.
- Validação de *query* anotada em consultas em tempo de inicialização.
- Suporte para XML de mapeamento de entidades.

3.1.8 Spring Boot

Atualmente, apesar de amplamente usado, o *Spring Framework* ainda é alvo de críticas a respeito do tempo e trabalho necessário para configurar o ambiente de desenvolvimento. A principal crítica feita ao *framework*, diz respeito à forma como o seu contêiner de injeção de dependências é configurado , via XML. Visando eliminar tal forma de configuração, foi inserido no *Spring*, a partir de sua versão 4.0, o micro *framework Spring Boot*.

A partir de seu uso, torna-se mais simples a forma de se criar aplicativos *Spring*, exigindo muito pouca configuração. Segundo a documentação oficial, o mesmo conta as principais características:

- Criar aplicações *Spring* autônomos.
- Incorporar *Tomcat*, *Jetty* ou *Undertow* diretamente, sem necessidade de implantar arquivos .war.

- Fornecer POMs de "iniciação" opinativo para simplificar a configuração do *Maven*.
- Configurar automaticamente o *Spring* sempre que possível.
- Fornece recursos prontos para produção, tais como métricas, exames e configuração exteriorizada.
- Sem geração de código e exigência de configuração XML.

3.1.9 Hibernate

O Hibernate é um *framework* para persistência de objetos em banco de dados relacionais para aplicações, principalmente em Java. Existe também a versão para o .Net o NHibernate. Ele permite o desenvolvimento de classes persistentes utilizando conceitos como: associação, herança, polimorfismo, composição e coleções. O *framework* possui sintaxe própria de expressões para acesso ao banco, chamado *Hibernate Query Language* (HQL)(HIBERNATE, 2015).

Sendo assim, o Hibernate é um provedor de JPA que implementa a JPA. Por ser amplamente usado comercialmente e possuir vasta documentação, é usado pelo *Spring Data* JPA como implementação padrão para camada de acesso a dados.

3.1.10 IntelliJ Idea

Desenvolvido pela *JetBrains*, e sendo utilizado por empresas como *Siemens*, *Google*, *UbiSoft* entre outras, o *IntelliJ* encontra-se atualmente na versão 14.1. A primeira versão do *IntelliJ IDEA* foi lançada em janeiro de 2001. Ele é um Ambiente de Desenvolvimento Integrado, projetado para ajudar no processo de codificação e suporta um grande número de diferentes estruturas e ferramentas. Ele trabalha com múltiplas linguagens de programação e inclui suporte completo para Java 8 e Java EE 7 (KROCHMALSKI, 2014).

Na Figura 4, é apresentada uma visão geral do IntelliJ IDEA.

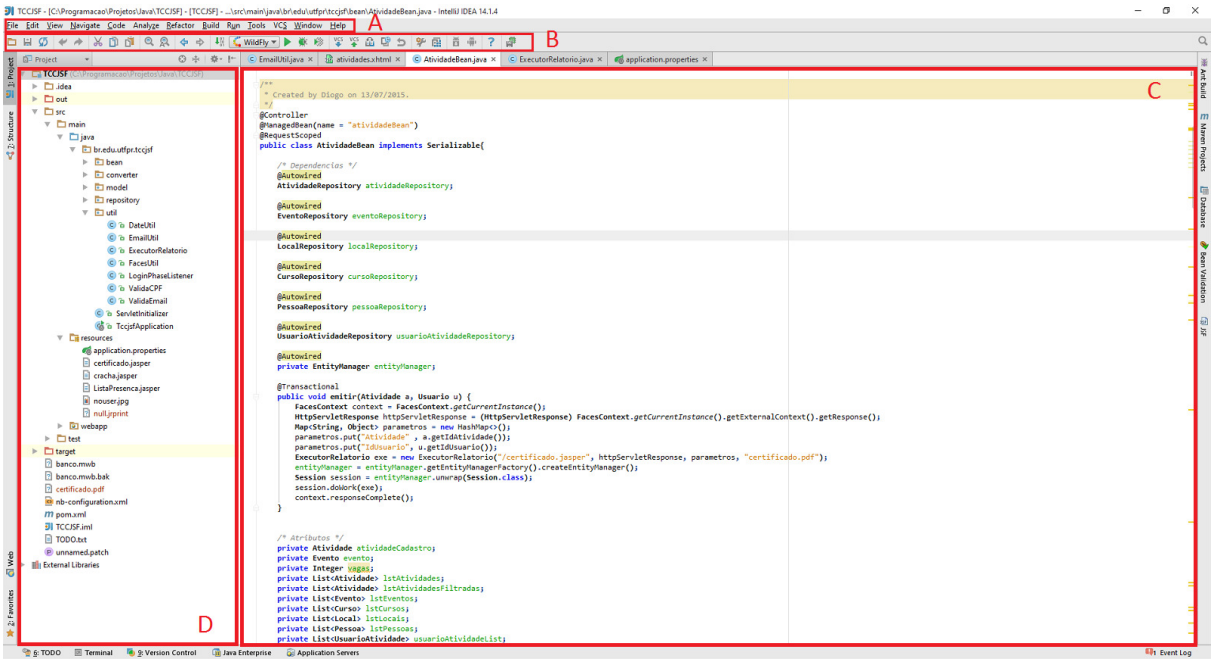

Figura 4 - Tela Inicial de Projetos do IntelliJ IDEA

Na Figura 4, pode-se identificar as seguintes áreas destacadas:

a) Barra de Menus - Local que do acesso as funcionalidades do sistema.

b) Barra de Ferramentas - Atalhos para as principais funcionalidades.

c) Área de Desenvolvimento - Área aonde são desenvolvidos e editados os códigos.

d) Explorador de Arquivos - Local aonde são exibidos os arquivos do projeto.

3.1.11 JBoss WildFly

Para Cmil *et al.* (2014), um servidor de aplicações é um ambiente de tempo de execução que fornece aplicativos com todos os componentes Java EE. O *Glassfish* é a implementação de referência patrocinado pela *Oracle*, mas a partir da versão 4 (criado para Java EE 7), perdeu seu suporte comercial. JBoss é uma divisão da *Red Hat*, que visa proporcionar um ecossistema *open source* para o desenvolvimento empresarial. Atualmente, a empresa apoia vários projetos (cerca de 100), e alguns deles são implementações de especificações Java EE. Os elementos corporativos são combinados em seu próprio servidor de aplicativos, o WildFly. Suas principais características são:

- *JBoss Application Server* substitui *JBoss AS*. A primeira versão do *JBoss Application Server* foi a 8.0, o qual foi baseado no *JBoss AS 7.1*. Para manter as coisas simples, a comunidade decidiu manter a mesma numeração.
- Ganhou a plataforma completa Java EE 7, o que significa que tem as mais recentes tecnologias Java. Desenvolvimento fácil, melhor segurança, melhor integração e gestão.
- Inicia-se em segundos. Todos os seus serviços iniciam-se juntos, mas apenas os que são necessários.
- *JBoss Application Server* tem apenas um arquivo de configuração, de modo que todas as suas configurações são centralizadas em um lugar (FUGARO, 2015).

A Figura 5 apresenta a tela inicial do painel de administração do servidor.

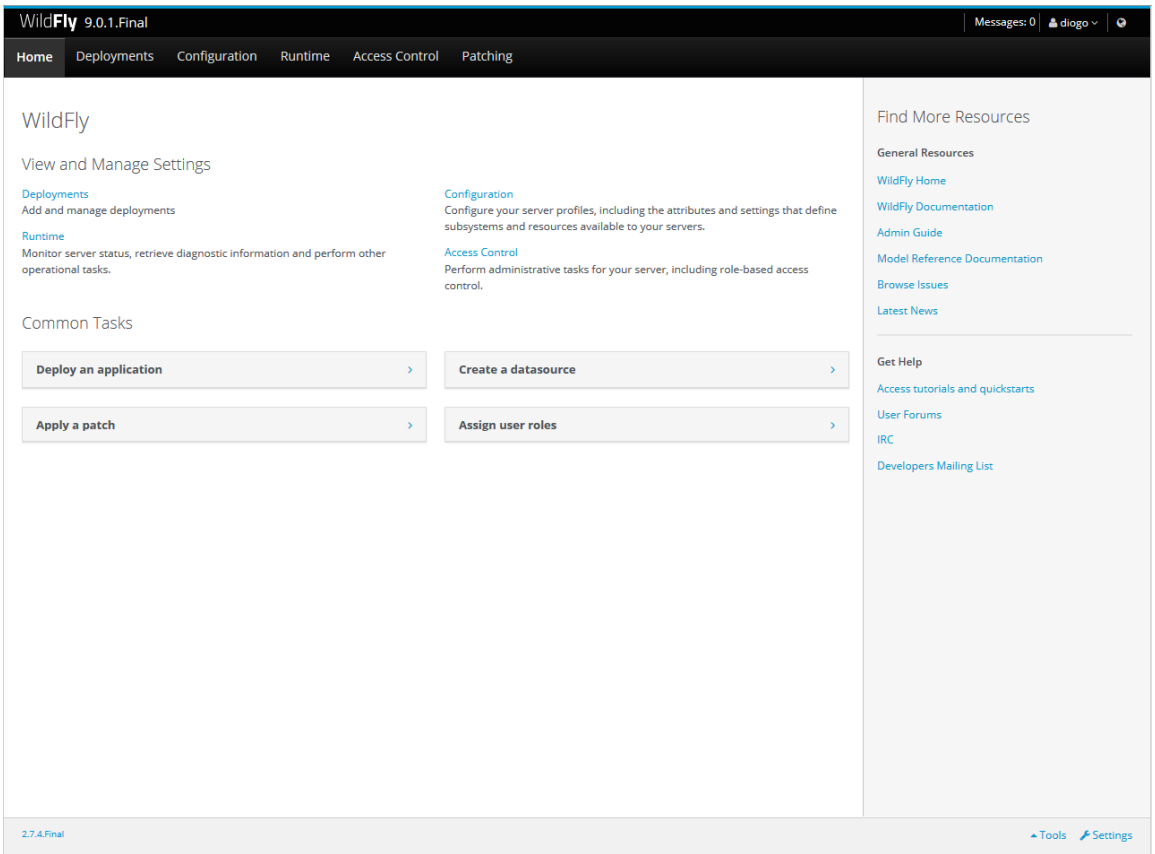

Figura 5 - Tela Inicial do Painel de Administração do WildFly

3.1.12 Visual Paradigm

Visual Paradigm for UML (VISUAL PARADIGM, 2010), é uma ferramenta *Computer Aided Software Engineering* (CASE) com várias opções para modelagem

com diagramas *Unified Modeling Language 2 (*UML) e também suporta *SysML*, diagramas de requisitos e Diagramas de Entidades e Relacionamentos (DER). A ferramenta tem um ambiente de trabalho que facilita a visualização e a manipulação do projeto de modelagem. É uma ferramenta de negócios e também suporta alterações no código fonte de algumas linguagens de programação, como C ++ e Java. A Figura 6 fornece uma visão da dela inicial da ferramenta.

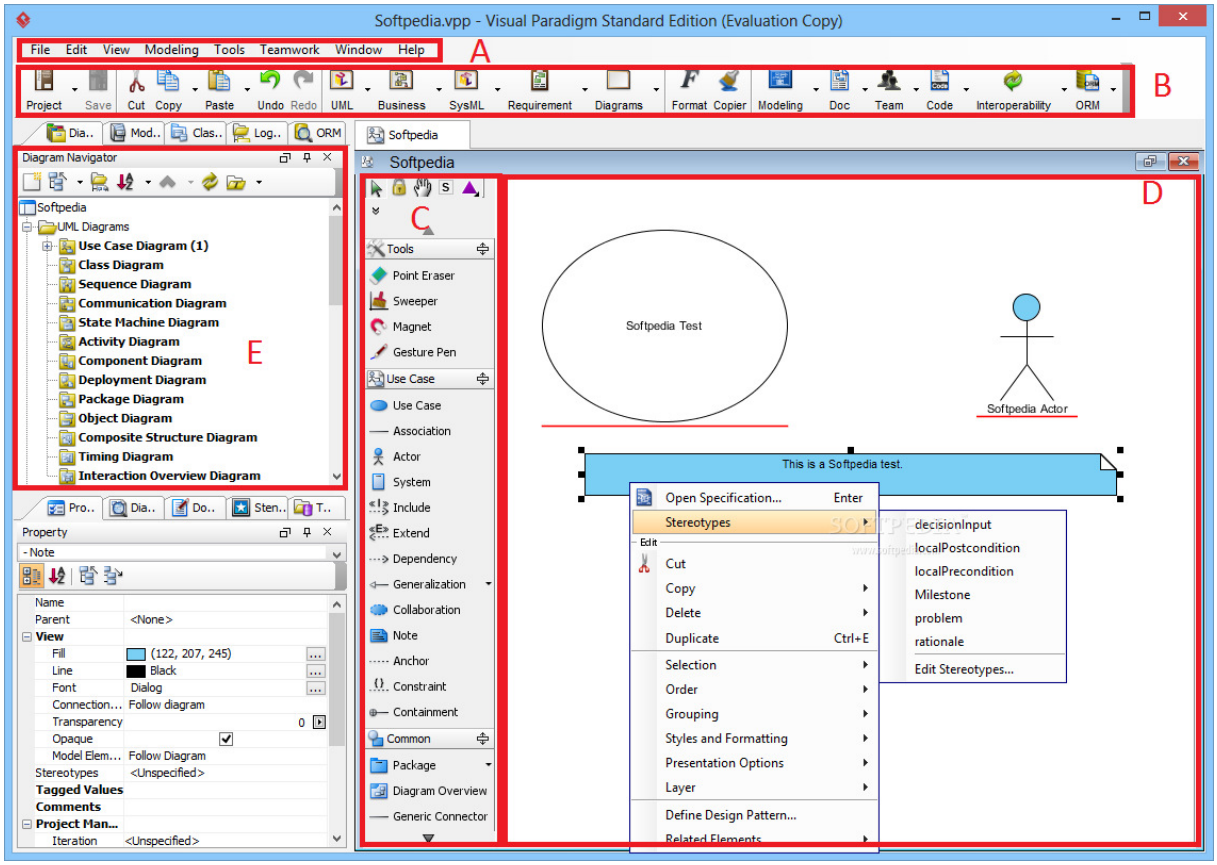

Figura 6 - Tela inicial do Visual Paradigm

Observando a Figura 6, estão algumas áreas destacadas:

a) Barra de Menu - Fornece acesso as funcionalidades do sistema.

b) Barra de Ferramentas - Atalhos para as principais funcionalidades do sistema.

c) Componentes e Ferramentas - Apresenta os elementos para criação de diagramas.

d) Área de criação de Diagramas - Local em que são criados e editados só diagramas.

e) Explorador de Diagramas - Área na qual são exibidos todos os diagramas desenvolvidos.

3.2 MÉTODO

Para o desenvolvimento do aplicativo para gerenciamento de eventos e a busca pelo conhecimento e familiarização com as tecnologias utilizadas, tais como o *Spring Framework* e o JavaEE, foram desenvolvidos diversos projetos com problemáticas similares as já encontradas no sistema proposto.

O levantamento de dados foi realizado utilizando técnicas de análise de requisitos, por meio de reuniões periódicas com o orientador do projeto. Nessas reuniões foram repassadas as principais necessidades da instituição e funcionalidades que deveriam se fazer presentes no sistema.

Os processos foram organizados com base no modelo sequencial linear como descrito em Pressman (2008), que consiste em análise, projeto, codificação e testes, tais processos sendo adequados ao decorrer do desenvolvimento.

A modelagem e desenvolvimento foram efetuados em iterações, inicialmente definindo os principais requisitos para obter-se uma visão geral do sistema. A seguir são descritos os processos utilizados:

a) Definição Requisitos:

Os requisitos iniciais foram levantados por meio de uma reunião inicial com o orientador, na qual foi apresentada a problemática do sistema proposto fornecendo uma visão geral do projeto, definindo as funcionalidades pretendidas. Tais requisitos foram complementados à medida que o sistema foi desenvolvido. Ao longo do desenvolvimento do projeto foram realizadas reuniões periódicas e troca de e-mails com o orientador possibilitando ajustes no projeto e definição de prazos para cada atividade.

b) Análise e Projeto do Sistema:

Por meio dos requisitos funcionais e não funcionais levantados foram definidos casos de uso do sistema, sendo documentados por meio de um diagrama. Com base em tal diagrama, foi possível definir o banco de dados e as classes do projeto, estando os mesmos documentados e descritos por meio de diagramas de classe e DER.

c) Implementação:

A implementação foi realizada utilizando a linguagem Java usando a especificação JavaEE, em conjunto com os *frameworks Java Server Faces* e *Primefaces* para *frontend*. Na parte *backend* foram utilizados os *frameworks Spring*

Boot e *Data JPA* integrado ao banco de dados MySQL e como servidor de aplicações foi utilizado o *JBoss WildFly*. Para tal desenvolvimento foi utilizado o IntelliJ IDEA por fornecer suporte à linguagem Java e à especificação JavaEE, bem como a todos os *frameworks* escolhidos para o sistema.

d) Testes:

Os testes foram realizados de forma informal pelo autor do trabalho, sem um plano de testes especifico, visando verificar o código desenvolvido e testes de interface e usabilidade.

4. RESULTADOS

O presente capítulo apresenta o sistema que foi desenvolvido como resultado deste trabalho, um sistema para gerenciamento de eventos.

4.1 APRESENTAÇÃO DO SISTEMA

O sistema desenvolvido realiza o controle dos processos de cadastro, gerenciamento, inscrições e emissão de certificados e relatórios necessários para a realização de um evento. Os dados serão armazenados em banco de dados MySQL.

O acesso ao sistema é de forma *online*, visto que se trata de uma aplicação desenvolvida para ambiente *Web*. O acesso é controlado por meio do cadastro do usuário previamente efetuado por meio de uma página específica para essa finalidade.

Os usuários são divididos em duas categorias, sendo elas: Usuários e Administradores. Por meio da categoria é efetuado o controle das permissões de acesso do sistema, visando maior segurança na aplicação. Desta maneira, é possível o controle de acesso às rotinas da aplicação e também permite trabalhar com apenas uma classe de usuário. Nessa classe são armazenados o Nome, E-mail, Cadastro de Pessoa Física (CPF), Instituição de Ensino, Curso, Registro Acadêmico, caso seja aluno da Universidade Tecnológica Federal do Paraná (UTFPR, e senha.

Um dos requisitos mais importantes do sistema diz respeito ao controle de acesso a atividades por parte do usuário. O cadastro de atividades está relacionado às classes de evento, local e responsável, e possui atributos de controle como restrito UTFPR (Sim ou Não), restrito a (Servidor ou Aluno) e curso. Esses atributos permitem que sejam controlados os tipos de atividades em que o usuário pode se inscrever, visto que o mesmo possui os atributos universidade, curso e tipo (Servidor ou Aluno). Um evento possui os atributos Nome, Data Inicial, Data Final e Descrição. A classe Local possui os atributos Descrição, Capacidade e Endereço (previamente cadastrado). Por sua vez, a classe Responsável, tem a responsabilidade de controlar o cadastro da pessoa responsável pela atividade, possuindo os seguintes atributos: Nome, E-mail, Telefone, Página Pessoal, Descrição, Foto e Assinatura.

Sendo assim, uma atividade, estando ligada às classes citadas acima, permite que sejam efetuados uma série de controles, como: número máximo de inscritos, data inicial e final, tanto para a realização quanto para inscrição do evento, emissão de certificados com assinatura automática, etc.

A aplicação permite aos usuários administradores realizarem o controle de presença em atividades via leitura de código de barras ou digitação do registro acadêmico ou CPF do usuário. O código de barras exigido para leitura por sua vez, encontra-se presente no crachá do aluno no caso de acadêmicos da UTFPR, ou pela emissão de um crachá contendo o CPF do usuário pelo próprio sistema. A rotina de controle de entradas em atividades possui como condicional o usuário estar matriculado na atividade selecionada.

O sistema permite que o próprio usuário faça a emissão do certificado assim que a atividade for finalizada. Também será possível o envio automático via e-mail de todos os certificados para os presentes em uma atividade.

4.2 MODELAGEM DO SISTEMA

Levando em consideração que hoje todo o controle do processo de eventos na UTFPR, Câmpus Pato Branco, é feito de maneira manual, o foco do projeto é de automatizar e gerenciar a realização de atividades pera instituição do início ao fim do processo, visando maior agilidade e segurança, tanto aos organizadores quanto aos participantes.

Segundo Rezende (2005), os requisitos funcionais são fundamentais e podem ser definidos como as funções ou atividades que o sistema faz (quando pronto) ou fará (quando em desenvolvimento). Devem ser definidos claramente e relatados explicitamente. Os requisitos de software são condições ou capacitações que devem ser contempladas pelo software, solicitada pelo cliente ou usuário para resolver um problema ou alcançar um objetivo.

Pela análise do processo, foi possível identificar os seguintes requisitos funcionais, os quais são apresentados no Quadro 1. RF significa Requisito Funcional

Identificação	Requisito	Descrição
RF001	Cadastrar Usuários / Administradores	Cadastro dos usuários que poderão ter acesso ao sistema, sendo usuário normal ou administrador.

RF002	Cadastrar Cidades	Cadastro das Cidades a serem utilizadas nos endereços do sistema.
RF003	Cadastrar Cursos / Departamentos	Cadastro dos cursos ou departamentos que ficarão ligados a usuários e atividades do sistema.
RF004	Cadastrar Universidades	Cadastro das universidades ligadas aos usuários e locais.
RF005	Cadastrar Endereços	Cadastro de endereços dos locais.
RF006	Cadastrar Locais	Cadastro de locais para atividades realizadas. Tais como salas, auditórios, centros de eventos, etc.
RF007	Cadastrar Pessoas	Cadastro de pessoas responsáveis pelas atividades realizadas.
RF008	Cadastrar Eventos	Cadastro de eventos que conterão as atividades realizadas.
RF009	Cadastrar Atividades	Cadastro de atividades realizadas pela instituição.
RF010	Efetuar Inscrição em Atividade	Possibilitar a inscrição nas atividades pelos usuários.
RF011	Efetuar Emissão de Certificados	Possibilitar a emissão de certificados pelo usuário para atividades as quais participar.
RF012	Efetuar Emissão de Certificados e envio por E-mail	Possibilitar a emissão de certificados para todos os usuários presentes em uma atividade e enviá-los por e-mail.
RF014	Efetuar Emissão de Crachá	O usuário poderá emitir seu próprio crachá com código de barras.
RF014	Efetuar Controle de Entradas em Atividades	Permitir que seja efetuado o registro de presença nas atividades por meio de leitura de código de barras do crachá ou digitação direta do registro acadêmico ou CPF.
RF015	Efetuar Manutenção do Perfil	O usuário poderá alterar informações de seu perfil, como telefone, e-mail, senha e foto do crachá.
RF016	Emitir Relatório de Lista de Presença	Permitir que seja emitido relatório listando os usuários matriculados que estiveram presentes na atividade.
RF017	Emitir Relatório de Lista de Presença x Curso	Permitir que seja emitido relatório listando os usuários matriculados que estiveram presentes na atividade, filtrando por curso específico.
RF018	Recuperação de Senha	Ao acessar a opção de recuperação de senha, será gerada uma nova senha, e enviada para o e-mail informado.

Quadro 1 - Requisitos Funcionais.

No Quadro 2, é apresentada a descrição do requisito funcional para cadastrar atividades.

RF01 Cadastrar Atividades	**Oculto ()**
Descrição: O usuário seleciona a opção cadastrar e é redirecionado para a página de inserção de atividades; informa então: nome, seleciona um evento vinculado à mesma,	

descrição, data e hora de início e fim; data de início e fim das inscrições; seleciona o local aonde será realizada a atividade; o responsável pela mesma, informa se a mesma emite certificado, se esta finalizada, número de vagas disponíveis, número de horas aula. Por fim é selecionada a opção Restrito UTFPR, se sim, informa também se é restrita a que tipo de usuários e de que curso/departamento. **Fontes:** Usuário administrador. **Usuário:** Usuário administrador. **Informações de Entrada:** Evento, responsável, local. **Informações de Saída:** Atividade. **Restrições lógicas:** A data de início e fim da atividade deve estar dentro do período do evento. Período de inscrição deve ser anterior à realização do evento. Número de vagas não deverá ultrapassar a capacidade máxima do local.

Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF 22.1 Informar data de início e final da Atividade	O usuário deve informar um período dentro do período de realização do evento.	Interface	( )	(X)
NF 22.2 Informar Período de inscrição	O usuário deve informar um período para inscrição na atividade, anterior a realização da atividade.	Interface	( )	(X)
NF 22.3 Informar número de Vagas	O usuário deve informar um número de vagas menor ou igual à capacidade do local selecionado.	Interface	( )	(X)

Quadro 2 - Requisito Funcional Cadastrar Atividades

No Quadro 3, estão os requisitos não funcionais identificados para o sistema. Os requisitos não funcionais explicitam regras de negócio, restrições ao sistema de acesso, por exemplo, requisitos de qualidade, desempenho, segurança e outros. RNF significa Requisito Não Funcional.

Identificação	Requisito	Descrição
RNF001	Segurança no cadastro de Administradores / Servidores	Somente um usuário administrador pode cadastrar novos usuários e atribuir o mesmo como administrador ou tipo servidor.
RNF002	Controle de segurança nas rotinas de Cadastro	Somente o usuário administrador terá acesso a cadastrar/editar todos os cadastros do sistema.
RNF003	Controle de segurança nas rotinas de emissão de relatórios, controle de entradas em atividades e envio de certificados via e-mail	Somente o usuário administrador terá acesso aos relatórios do sistema, rotina de entradas em atividades e envio de certificados via e-mail.
RNF004	Controle de segurança em Inscrição em Atividades	O usuário poderá se inscrever apenas em atividades abertas, ou restritas somente a sua universidade/curso.

RNF05	Controle de exclusão de Usuários	Não podem ser excluídos usuários que já tenham relacionamento com atividades ou estiverem utilizando o sistema.
RNF006	Controle de exclusão de Cidades	Não podem ser excluídos cidades que já tenham relacionamento com universidades.
RNF007	Controle de exclusão de Universidades	Não podem ser excluídas universidades.
RNF008	Controle de exclusão de Endereços	Não podem ser excluídos endereços que já possuam ligação com locais.
RNF009	Controle de exclusão de Locais	Não podem ser excluídos locais que já possuam ligação com atividades.
RNF010	Controle de exclusão de Pessoas	Não podem ser excluídas pessoas que já possuam ligação com atividades.
RNF011	Controle de exclusão de Eventos	Não podem ser excluídos eventos que já tenham relacionamento com atividades.
RNF012	Controle de exclusão de Atividades	Não podem ser excluídos atividades que já possuam usuários inscritos.
RNF013	Controle de exclusão de Usuário / Atividades	Uma vez inscrito em uma atividade, não é possível cancelar tal inscrição.

Quadro 3 - Requisitos não Funcionais

Os requisitos foram organizados em forma de casos de usos, apresentando as funcionalidades do sistema, em linguagem simplificada e de fácil entendimento. Esse diagrama identifica os atores (usuários do sistema), bem como as opções que serão disponibilizadas pelo sistema. Na Figura 7, é apresentado o diagrama mencionado.

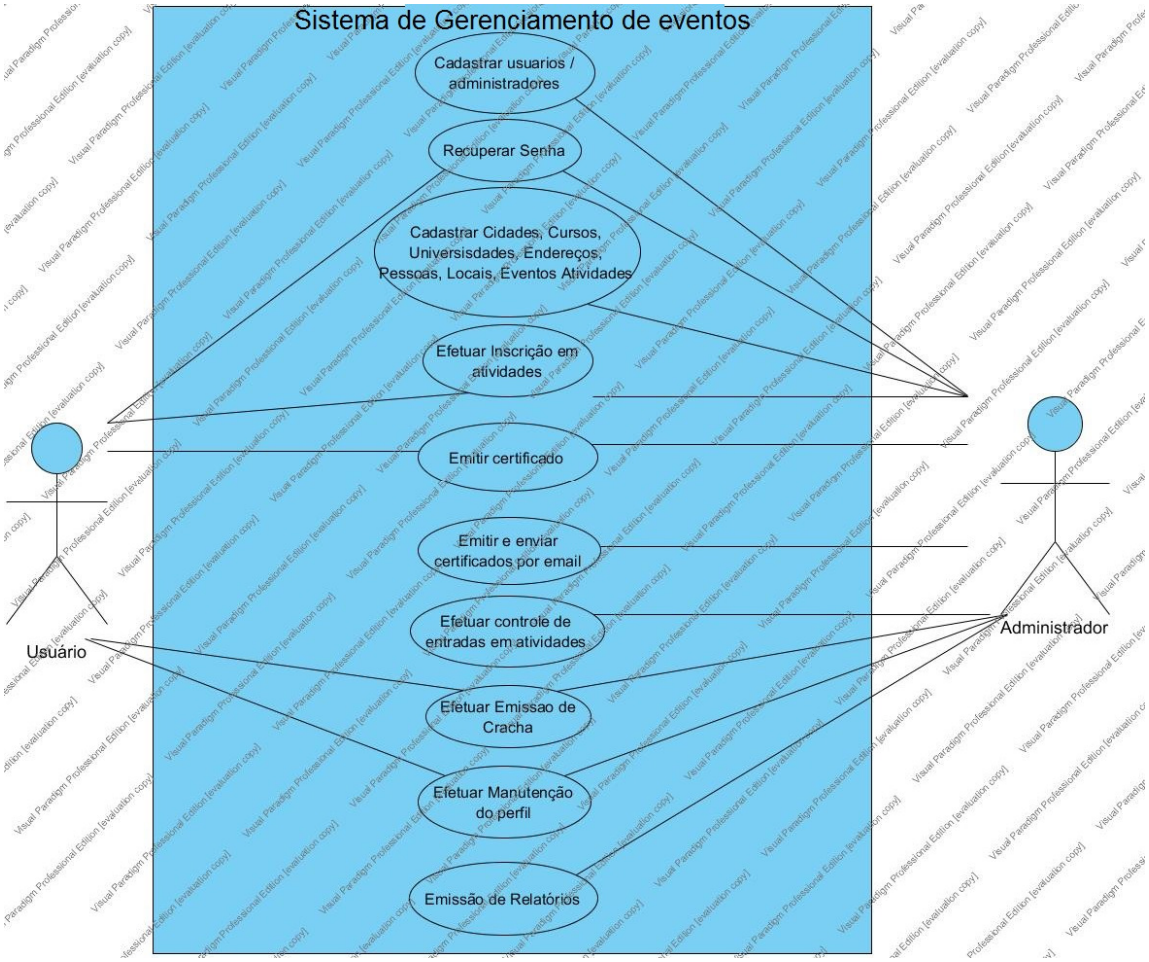

Figura 7 - Casos de Uso do Sistema

Já na Figura 8, é apresentado o diagrama de entidades relacionais, gerado pela ferramenta MySQL *Workbench*.

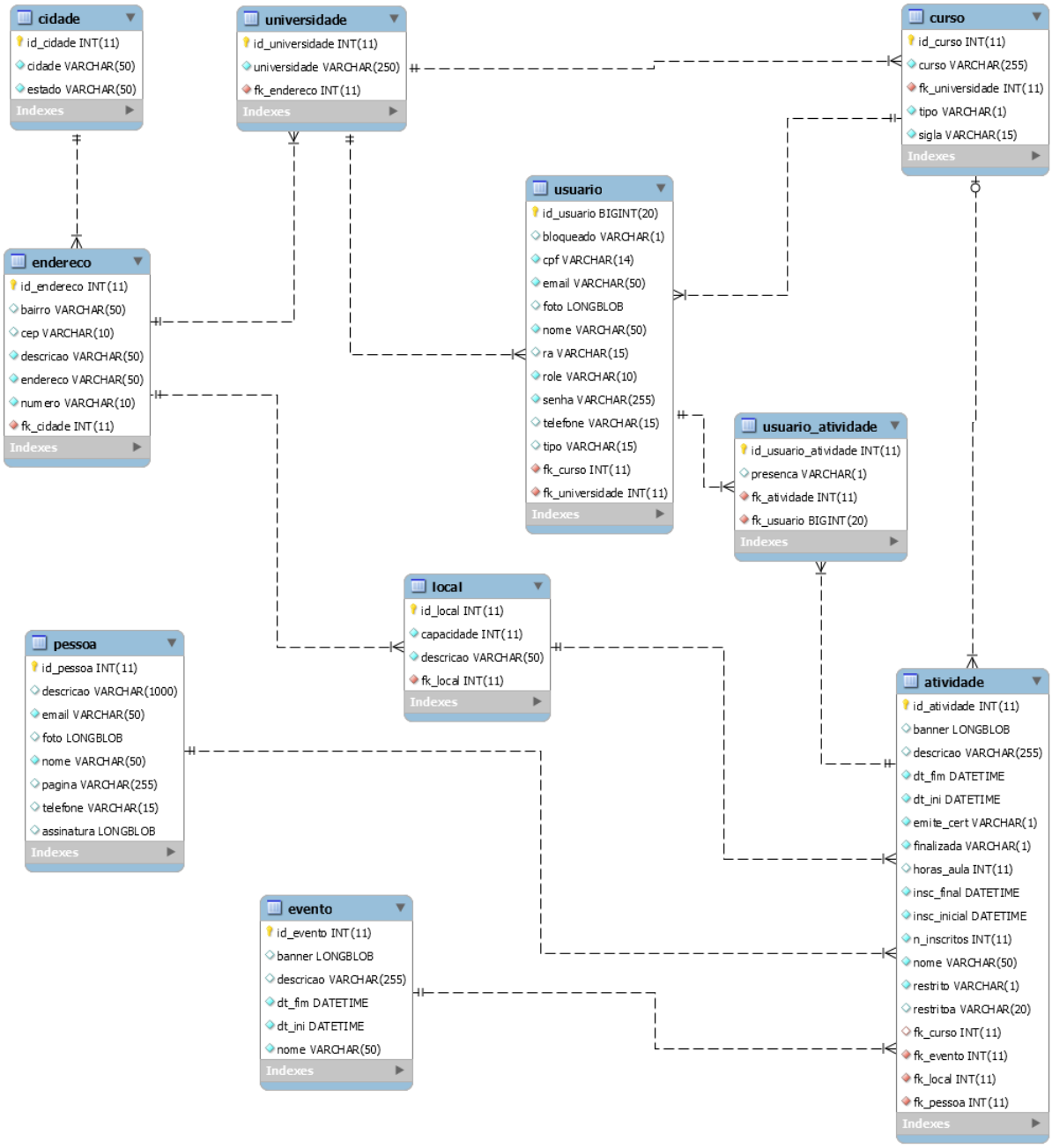

Figura 8 - Diagrama de Entidades Relacionais

A seguir são apresentadas as tabelas do sistema em detalhe, com uma descrição dos principais campos e suas finalidades. O Quadro 4 descreve os campos do cadastro de cidades.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_cidade	Integer	*		*	Código da cidade.
cidade	Varchar (50)			*	Descrição da cidade.
estado	Varchar (50)			*	Sigla do Estado.

Quadro 4 - Tabela Cidade

Os endereços estão relacionados a universidades e locais. Esse relacionamento é utilizado para obter o endereço da atividade ao selecionar um local. Em um exemplo simples, tem-se o endereço com o nome UTFPR, contendo os dados da instituição. Por sua vez, quando o cadastro de um local, por exemplo, Mini Auditório da UTFPR, será selecionado o endereço da UTFPR, fazendo a ligação local/endereço. O Quadro 5 descreve os campos do cadastro de endereços, complementada pela ligação com a tabela de cidades.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_endereço	Integer	*		*	Código do endereço.
bairro	Varchar (50)				Descrição do bairro.
cep	Varchar (10)				CEP do endereço.
descrição	Varchar (50)			*	Descrição do endereço.
endereço	Varchar (50)			*	Endereço (Rua).
número	Varchar (10)			*	Número do endereço.
fk_cidade	Integer		*	*	Código da cidade.

Quadro 5 - Tabela Endereço

Tais locais, no contexto do sistema desenvolvido, seriam salas, auditórios, teatros, centros de eventos e todo tipo de local que possa receber uma atividade. Já o Quadro 6, apresenta a tabela de locais, contendo a ligação com a tabela de endereços.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_local	Integer	*		*	Código do local.
capacidade	Integer			*	Capacidade do local.
descrição	Varchar (10)			*	Descrição do local.
fk_endereço	Integer		*	*	Código do endereço.

Quadro 6 - Tabela Locais

O cadastro da universidade leva seu endereço, que é cadastrado separadamente. Universidades são utilizadas no cadastro de usuários possibilitando uma série de restrições de acesso a atividades. O Quadro 7 apresenta a tabela de universidades.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_universidade	Integer	*		*	Código da universidade.
universidade	Varchar (250)			*	Descrição da universidade.
fk_endereço	Integer		*	*	Código do endereço.

Quadro 7 - Tabela Universidades

A tabela de cursos possui o campo Tipo, para especificar se é um curso ou um departamento ligado a uma universidade em particular, o qual também é utilizado como forma de restrição a inscrições em eventos por usuários. O Quadro 8 apresenta a tabela de cursos.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_curso	Integer	*		*	Código do curso.
curso	Varchar (250)			*	Descrição do curso.
fk_universidade	Integer		*	*	Código da universidade.
tipo	Varchar (1)			*	Tipo do curso (curso ou departamento).
sigla	Varchar (15)			*	Sigla do curso.

Quadro 8 - Tabela Cursos

Na tabela de Usuários, merecem destaque os campos: Bloqueado, Registro Acadêmico, Role e Tipo. O campo bloqueado, quando marcado com S, bloqueia o acesso do usuário ao sistema. O RA (Registro acadêmico) armazena o código do aluno, exigido somente caso usuário selecione a universidade UTFPR. O atributo role, armazena informações de direitos de acesso do usuário. Serão armazenados o valor “USER” para usuários normais e “ADM” para usuários administradores. O campo Tipo armazena por sua vez, o tipo do usuário, podendo ele ser do tipo “Aluno” ou “Servidor”. Esses campos possibilitam validações de controle de acessos a rotinas especificas do sistema e inscrições em atividades. O Quadro 9 apresenta a tabela de usuário.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_usuário	Integer	*		*	Código do usuário.
bloqueado	Varchar (1)				Status do usuário (S/N).
cpf	Varchar (14)			*	CPF do usuário.

e-mail	Varchar (50)			*	E-mail do usuário.
foto	Longblob				Foto do usuário.
nome	Varchar (50)			*	Nome do usuário.
ra	Varchar (15)				Registro Acadêmico.
role	Varchar (10)			*	Permissão (USER, ADM).
senha	Varchar (255)			*	Senha do usuário.
telefone	Varchar (15)				Telefone do usuário.
tipo	Varchar (15)				Tipo (Aluno/Servidor).
fk_curso	Integer		*	*	Código do curso.
fk_universidade	Integer		*	*	Código da universidade.

Quadro 9 - Tabela Usuário

Tendo em mente o contexto da aplicação desenvolvida, a tabela de Pessoas é utilizada para armazenar pessoas responsáveis por eventos, por exemplo, palestrantes, professores, servidores específicos, etc. O Quadro 10 apresenta a tabela de pessoas.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_pessoa	Integer	*		*	Código da pessoa.
descrição	Varchar (1000)				Descrição da pessoa.
e-mail	Varchar (50)			*	E-mail da pessoa.
foto	Longblob				Foto da pessoa.
nome	Varchar (50)			*	Nome da pessoa.
página	Varchar (255)				Página pessoal (Facebook, Lattes).
assinatura	Longblob				Imagem contendo a assinatura.
telefone	Varchar (15)				Telefone da pessoa.

Quadro 10 - Tabela Pessoa

Um evento pode conter uma série de atividades a serem realizadas, como por exemplo, encontros e convenções nas quais são realizados palestras, cursos, oficinas e outros. Neste caso, tem-se apenas um evento, no qual são realizados uma

série de atividades. Sendo assim, no Quadro 11 é apresentada a tabela de eventos do sistema.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_evento	Integer	*		*	Código do evento.
banner	Longblob				Imagem banner do evento.
descrição	Varchar (255)				Código do endereço.
dt_ini	Datetime			*	Data e hora início do evento.
dt_fin	Datetime			*	Data e hora final do evento.
nome	Varchar (50)			*	Nome do evento.

Quadro 11 - Tabela Evento

O Quadro 12 apresenta a tabela atividades.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_atividade	Integer	*		*	Código da atividade.
banner	Longblob				Imagem banner da atividade.
descrição	Varchar (255)				Descrição da atividade.
dt_ini	Datetime			*	Data e hora início da atividade.
dt_fin	Datetime			*	Data e hora final da atividade.
emite_cert	Varchar (1)			*	Atividade emite certificado (S/N).
finalizada	Varchar (1)			*	Atividade finalizada (S/N).
horas_aula	Integer				Número de horas aula para geração de certificado.
insc_incial	Datetime			*	Data inicial para inscrições.
insc_final	Datetime			*	Data final para inscrições.
n_inscritos	Integer			*	Número máximo de inscritos.
nome	Varchar (50)			*	Nome da atividade.
restrito	Varchar (1)			*	Atividade restrita a UTFPR (S/N).
restriroA	Varchar (20)				Restrita a apenas (Alunos/Servidores).
fk_curso	Integer		*		Restrita a alunos ou servidores do curso/departamento.
fk_evento	Integer		*	*	Código do evento ao qual a atividade pertence.

fk_local	Integer		*	*	Código do local em que a atividade será realizada.
fk_pessoa	Integer		*	*	Código da pessoa responsável.

Quadro 12 - Tabela Atividades

Visto as tabelas anteriores, fez-se a necessidade de uma tabela que armazene os alunos por atividade, representando as inscrições efetuadas nas atividades disponibilizadas. Nessa tabela está presente o campo presença, que ao ser inserido o registro, representando uma inscrição, é marcado automaticamente com o valor N. Quando o registro de entradas em atividades for confirmado a presença do usuário da atividade, o mesmo é alterado para S. O Quadro 13 ilustra a tabela usuário x atividade.

Nome do Campo	Tipo	PK	FK	Não nulo	Descrição
id_usuário_atividade	Integer	*		*	Código do usuário atividade.
presença	Varchar (1)				Presença na atividade (S/N).
fk_atividade	Integer		*	*	Código da atividade.
fk_usuário	Integer		*	*	Código do usuário.

Quadro 13 - Tabela Usuário - Atividade

4.3 DESCRIÇÃO DO SISTEMA

O sistema foi desenvolvido utilizando a linguagem Java seguindo a especificação Java EE, como descrito no item 3.1.1. Como resultado, foi desenvolvida uma solução para gerenciamento de eventos, os requisitos foram feitos com base nas necessidades de uma instituição de ensino, no caso deste trabalho, a UTFPR Câmpus Pato Branco.

O acesso inicia-se pela tela de *login*, na qual são exigidas informações de E-mail e senha para acesso. Ambos os campos são de informação obrigatória e é necessário que o usuário já esteja previamente cadastrado no sistema. Na tela de *login* também são apresentados dois links. Sendo um para recuperação de senha de usuário, em caso de esquecimento da mesma, e outro para a tela de cadastro de usuário. A Figura 9 exibe a tela de *login*.

Figura 9 - Tela de Login

Acessando o endereço de recuperação de senha, o usuário é redirecionado a página de geração de nova senha, na qual é apresentado um formulário exigindo que seja informado o e-mail do usuário. Clicando no botão Gerar Senha, uma nova senha é gerada para o usuário e enviada para e e-mail informado. Ao fim do processo é exibida uma mensagem de sucesso ou erro no canto superior direito da tela. A Figura 10 apresenta o formulário para geração de nova senha.

Figura 10 - Formulário de Geração de nova Senha

Em todas as telas do sistema, são exibidas mensagens de sucesso ou erro ao executar funções. A apresentação de tais mensagens é padrão, sendo alterado

apenas o texto das mesmas. A Figura 11 ilustra um exemplo de mensagens do sistema.

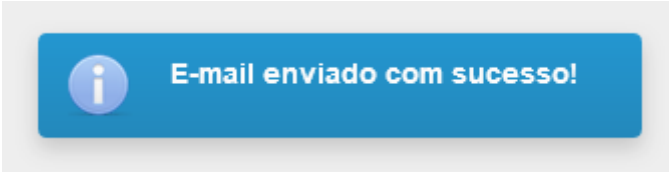

Figura 11 - Exemplo de Mensagem do Sistema

A partir da tela de login é possível também o acesso à página de cadastro de usuário. No formulário de cadastro são exigidas as informações de nome, e-mail, CPF, telefone, instituição de ensino, curso, registro acadêmico caso o usuário selecione como universidade a UTFPR, senha e confirmação de senha. Todos os campos, com exceção do telefone, são de informação obrigatória. Ao efetuar o cadastro, as informações são gravadas na tabela usuário do banco de dados. Por padrão os campos de controle tipo e *role* são populados com os valores “Aluno” e “USER” respectivamente.

Esses valores são adotados para evitar que o aluno tenha acesso aos cadastros e relatórios do sistema. As restrições de acesso são efetuadas pela validação de tais campos. Usuários administradores ou servidores devem ser cadastrados internamente no sistema, por rotina específica a ser apresentada posteriormente. A Figura 12 mostra a tela de cadastro de usuário.

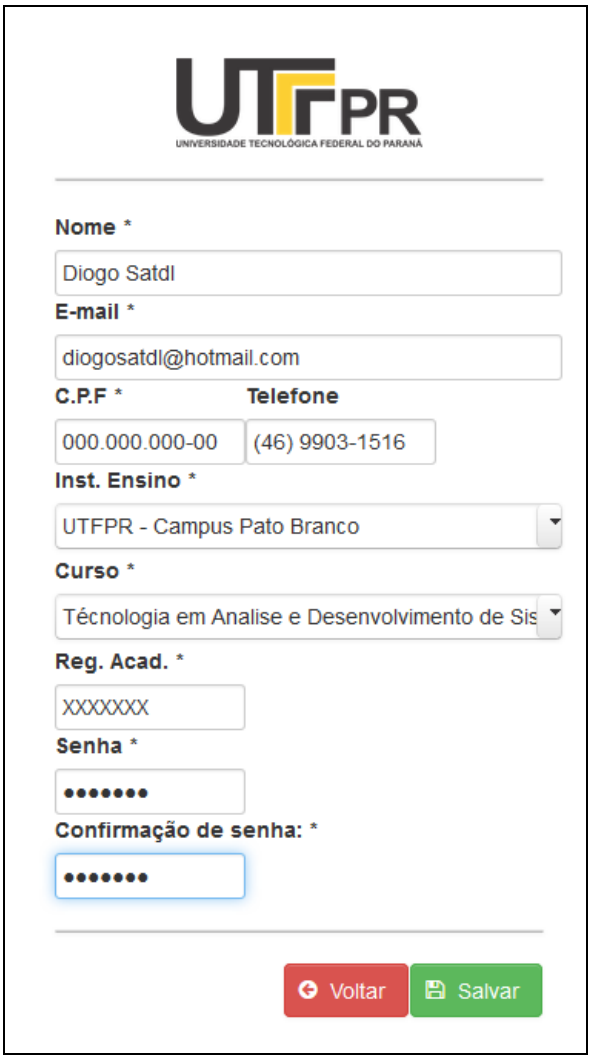

Figura 12 - Formulário de Cadastro de Usuários

Na Figura 13 é apresentada a tela inicial do sistema, fazendo acesso com usuário administrador. Pode-se observar na imagem os menus do sistema, os quais são carregados de maneira dinâmica, conforme o direito de acesso do usuário.

Figura 13 - Tela inicial do Sistema - Usuário Administrador

A Figura 14 demonstra a tela inicial quando acessada por um usuário normal, ou seja, sem direito de acesso de administrador. Observa-se que os menus Cadastros, Registro de Entradas e Relatórios são ocultados do usuário. O acesso também é bloqueado caso o usuário tente acessar diretamente a URL de alguma dessas páginas.

Figura 14 - Tela inicial do Sistema - Usuário Normal

4.3.1 PÁGINAS DE CADASTRO PADRÃO

O menu Cadastros possui os seguintes submenus: Usuários, Cidades, Universidades, Cursos/Departamentos, Endereços, Locais, Pessoas, Eventos, Atividades. Abaixo, a Figura 15 demonstra o menu de Cadastros.

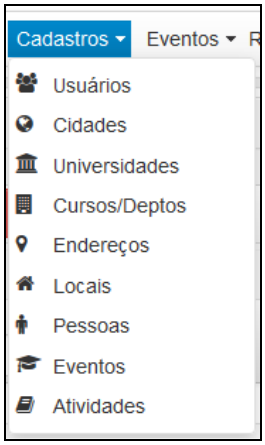

Figura 15 - Menu Cadastros e seus Sub Menus

Com exceção das páginas de cadastro de pessoas, eventos e atividades, as telas de manutenção de cadastros do sistema, foram desenvolvidas utilizando um modelo padrão e possuem funcionamento semelhante, apresentando uma listagem dos cadastros já efetuados e opções de Cadastro, Exclusão e Edição. Salvas as exceções mencionadas acima, o restante das páginas possui o mesmo funcionamento básico, somente sendo alterado o cadastro em questão, e serão omitidas desse capítulo para evitar que o texto se torne demasiado extenso. Será utilizada a rotina de manutenção de usuários para exemplificar o sistema de manutenção cadastros da aplicação.

Ao acessar o menu Usuários, o administrador do sistema é redirecionado para a página de cadastro de usuários. Nesta tela, é apresentada uma listagem dos usuários cadastrados no sistema em forma de *grid*, contendo ordenação e opções de filtragem e paginação, semelhante a uma aplicação *desktop*. Na mesma tela, também são apresentadas as opções Cadastrar, Editar e Excluir, como pode ser observado na Figura 16.

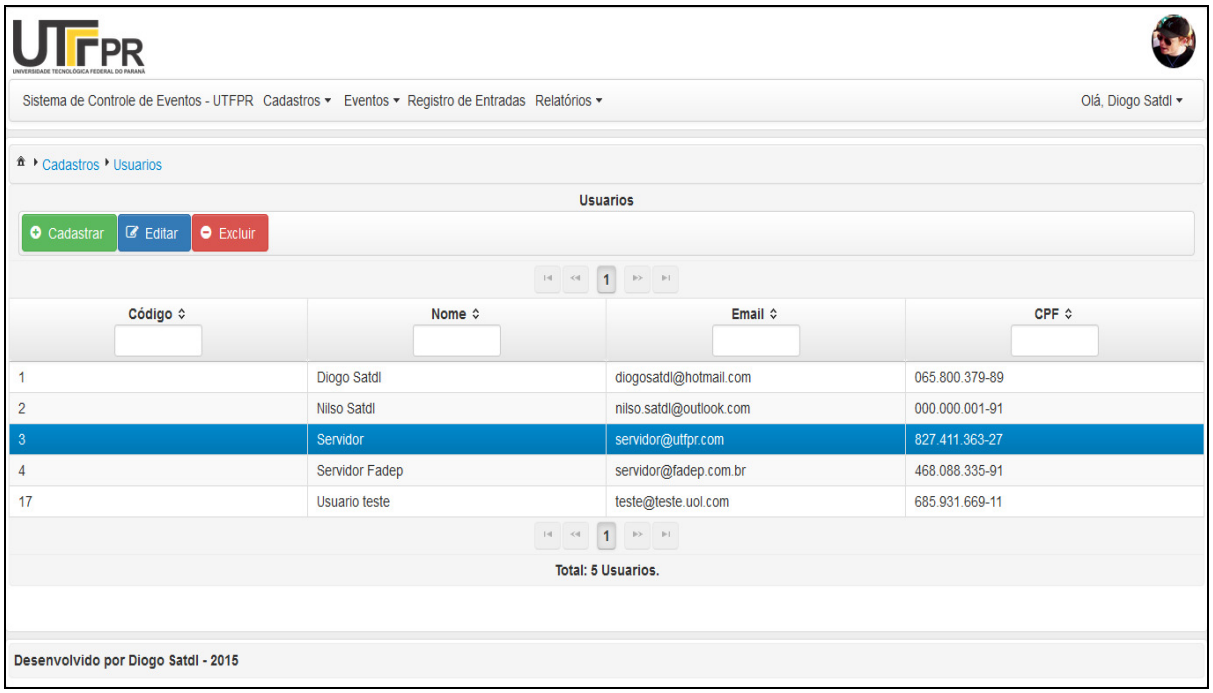

Figura 16 - Tela de Cadastro de Usuários

Ao acessar a opção cadastrar é exibida uma caixa de diálogo, contendo o formulário para cadastro de usuários. Esse formulário contém os mesmos campos que a tela de cadastro de usuários inicial, com adição dos campos Acesso, Bloqueado e Tipo. Campos esses que dizem respeito ao tipo de acesso do usuário (Normal ou Administrador), se o mesmo possui o acesso ao sistema bloqueado (Sim ou Não) e tipo do usuário (Aluno ou Servidor), respectivamente. Tais campos foram adicionados nesta tela, pois é por meio dela que deve ser feito o cadastro de outros usuários administradores ou servidores. Ao fazer ou finalizar o cadastro do usuário, clicando no botão Gravar, a caixa é fechada, uma mensagem de confirmação é exibida e o registro é automaticamente adicionado ao *grid* via AJAX.

A Figura 17 apresenta a caixa de diálogo utilizada no cadastro de usuários. Ao acessar a opção editar, a mesma caixa de diálogo é exibida, com a diferença de que os campos do formulário são exibidos com os dados do registro selecionado no *grid*. Quando do clique do botão, sem que esteja selecionado algum registro no *grid*, uma mensagem de erro é apresentada. Após alterar as informações e clicar no botão gravar, novamente a caixa é fechada, uma mensagem é apresentada ao usuário e o *grid* é atualizado via AJAX.

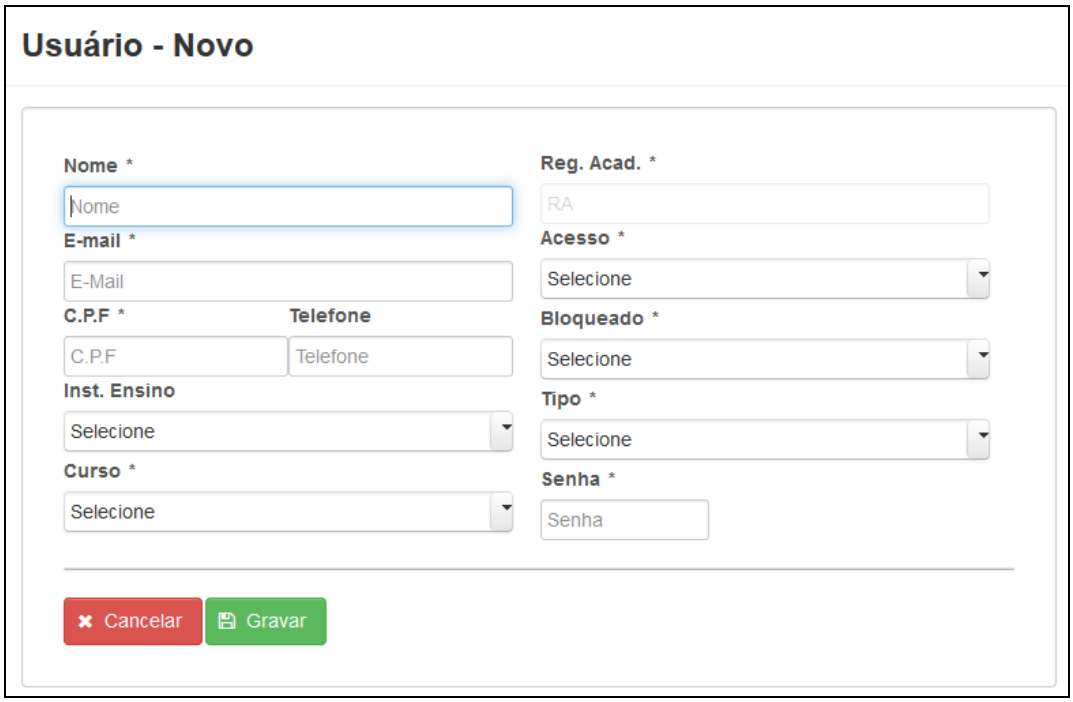

Figura 17 - Caixa de diálogo Cadastro de Usuário

A Figura 18 exibe a caixa de diálogo de edição de usuário.

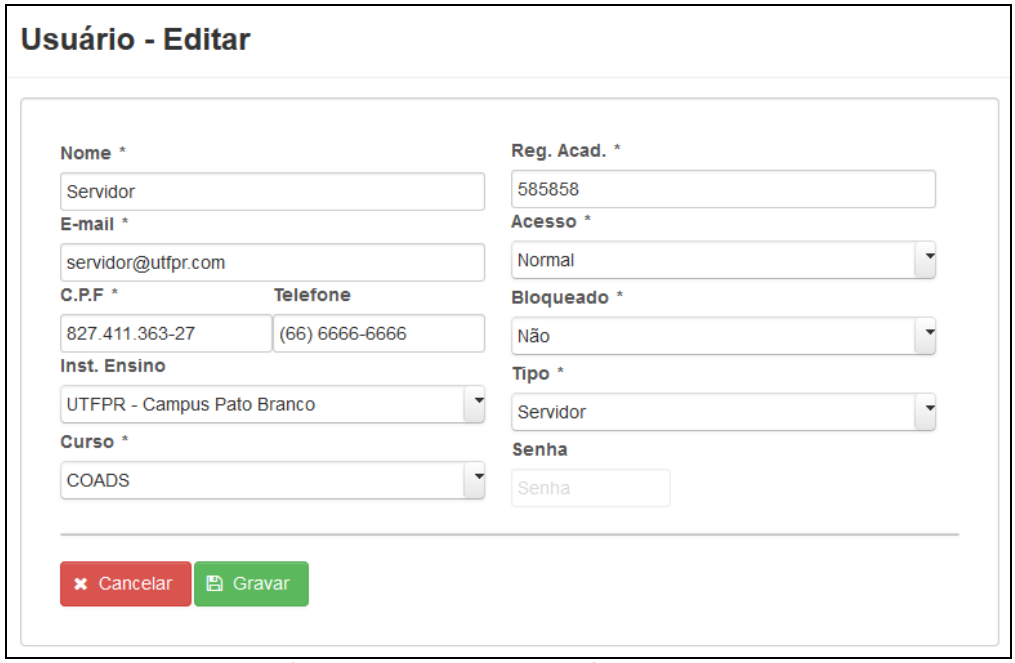

Figura 18 - Caixa de Diálogo de Edição de Usuários

Efetuando o clique na opção excluir, o mesmo se repete igualmente a opção editar, com a exceção que os campos do formulário são apresentados bloqueados. A caixa de diálogo serve como forma de confirmação para exclusão do usuário. A exclusão apresenta algumas restrições, sendo elas: o usuário não pode excluir seu próprio usuário e também não podem ser excluídos usuários que já possuam

inscrições em atividades, devido a vínculos entre as tabelas usuário e atividade. A Figura 19 apresenta a caixa de diálogo de exclusão de usuários.

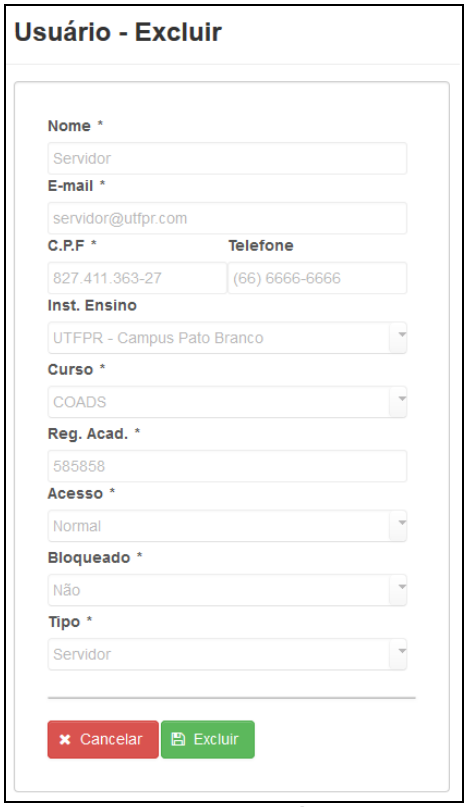

Figura 19 - Caixa de Diálogo de exclusão de Usuários

A Figura 20 apresenta exemplos de mensagens de sucesso e erro que são apresentadas durante a manutenção de usuários.

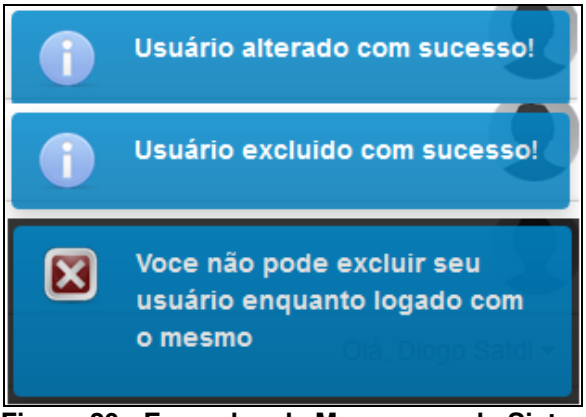

Figura 20 - Exemplos de Mensagens do Sistema

4.3.2 CADASTRO DE PESSOAS

Dentro do contexto da aplicação desenvolvida, o cadastro de pessoas fica responsável por gerenciar o controle dos responsáveis pelas atividades, como professores, palestrantes, etc. Esse cadastro possui algumas particularidades em comparação às outras telas de cadastro que são as opções de Alterar foto e Alterar Assinatura. O funcionamento desse cadastro é semelhante aos demais do sistema, com a tela principal contendo um *grid* de listagem das pessoas previamente cadastradas, opções de cadastrar, editar e excluir. Na Figura 21 é exibida a tela de controle de pessoas.

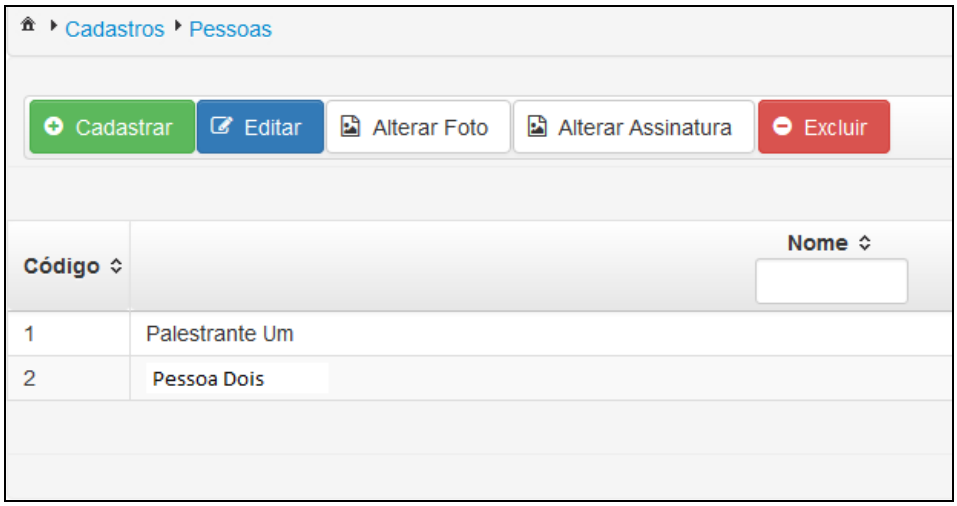

Figura 21 – Tela de controle de pessoas

Assim como os botões Editar e Excluir, as duas opções mencionadas necessitam que um cadastro esteja selecionado no *grid*. Ao clicar em um dos botões, uma caixa de diálogo é exibida, contendo as opções Selecionar, Cancelar e Enviar. A opção selecionar abre uma janela para seleção de uma imagem no computador no formato JPG ou JPEG com no máximo 2 megabytes. Após selecionar a imagem e clicar na opção enviar, a imagem é armazenada no banco de dados. A opção cancelar fecha a caixa de diálogo. Ao abrir novamente a opção para alteração de Foto ou Assinatura, a imagem enviada anteriormente é exibida na caixa de diálogo. As opções de alteração de imagem possuem o mesmo funcionamento. A imagem da assinatura é utilizada para emissão do certificado, o qual será detalhado posteriormente. As Figuras 22 e 23 apresentam respectivamente as caixas de diálogo de alteração de foto e assinatura.

Figura 22 - Caixa de Diálogo de alteração de Foto no Cadastro de Pessoas

Figura 23 - Caixa de Diálogo de alteração de Assinatura no Cadastro de Pessoas

4.3.3 CADASTRO DE EVENTOS

Seguindo o padrão do modelo de páginas já apresentado anteriormente, a tela de cadastro de eventos, possui as mesmas opções padrão, sendo elas Cadastrar, Editar e Excluir. Apresentando caixas de diálogo para efetuar as ações e tendo apenas a restrição de não poderem ser excluídos eventos que já possuem atividades relacionadas a ele. A particularidade que merece destaque neste caso fica por conta da tela de programação do evento que fica ligada ao *grid* de eventos cadastrados. A Figura 24 exemplifica a funcionalidade mencionada.

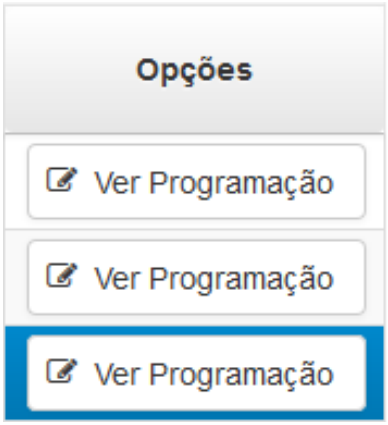

Figura 24 - Funcionalidade Ver Programação

A opção exibida na Figura 24 é dada na forma de uma coluna no *grid*, contendo um botão que leva para a página que exibe a programação do evento. A página exibe todas as atividades em aberto que estão ligadas ao evento selecionado. Essa funcionalidade está acessível somente a usuários administradores, porém, a mesma página pode ser acessada por usuários normais por meio da listagem de eventos, listando apenas os eventos que possuem atividades em aberto às quais usuário tem acesso a se inscrever. A listagem de eventos acessível a usuários sem direitos de acesso de administrador será mencionada posteriormente. A Figura 25 mostra a página de programação para um evento selecionado.

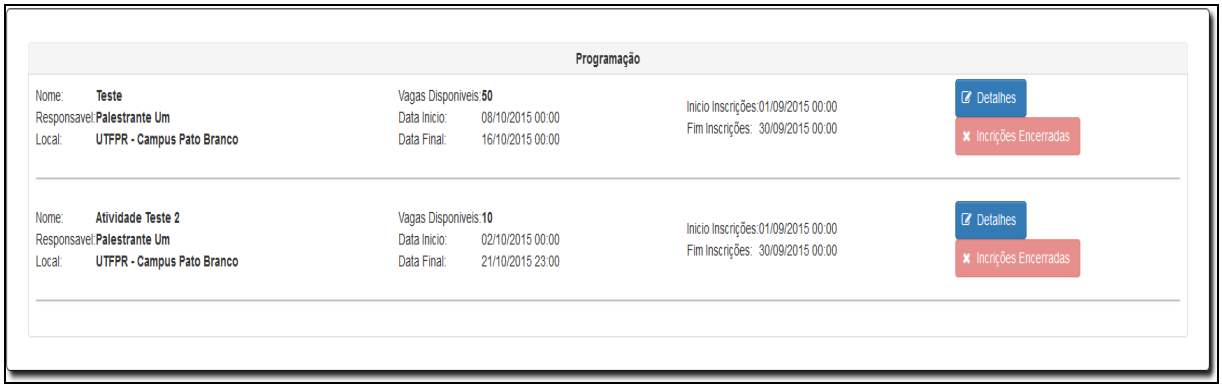

Figura 25 - Tela de Programação por Evento

A partir da tela de programação do evento, é possível acessar a página contendo os detalhes da atividade e a página que efetua a inscrição na mesma. Essas telas serão descritas em detalhes no subcapítulo 4.3.5 visto que se tratam das mesmas páginas para usuários normais e administradores, apenas mudando o caminho pelo qual são acessadas. Quando o usuário já estiver inscrito no evento, o

botão que faz o redirecionamento para a página de inscrição é ocultado e em seu lugar é exibido uma mensagem avisando que o usuário já está inscrito na atividade. Na mesma página, é exibido um botão em forma de rótulo com o texto inscrições encerradas, caso o período de inscrições esteja encerrado. Caso todas as atividades do evento selecionado já estiverem finalizadas, é exibida apenas uma mensagem informativa no lugar da listagem, a qual é exibida na Figura 26.

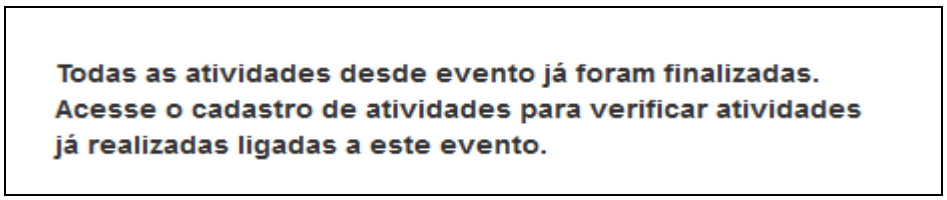

Figura 26 - Mensagem para Programação encerrada

4.3.4 ATIVIDADES E ENVIO CERTIFICADO VIA EMAIL

O cadastro de atividades foi implementado sem a utilização de caixas de diálogo, pois esse cadastro possuir uma quantidade maior de campos quando comparado com as telas apresentadas anteriormente, o que impossibilitou colocá-los dentro de apenas uma caixa de diálogo. Sendo assim, essa rotina foi implementada utilizando redirecionamento a uma página específica de cadastro, contendo todos os campos a serem informados, como pode ser observado na Figura 27.

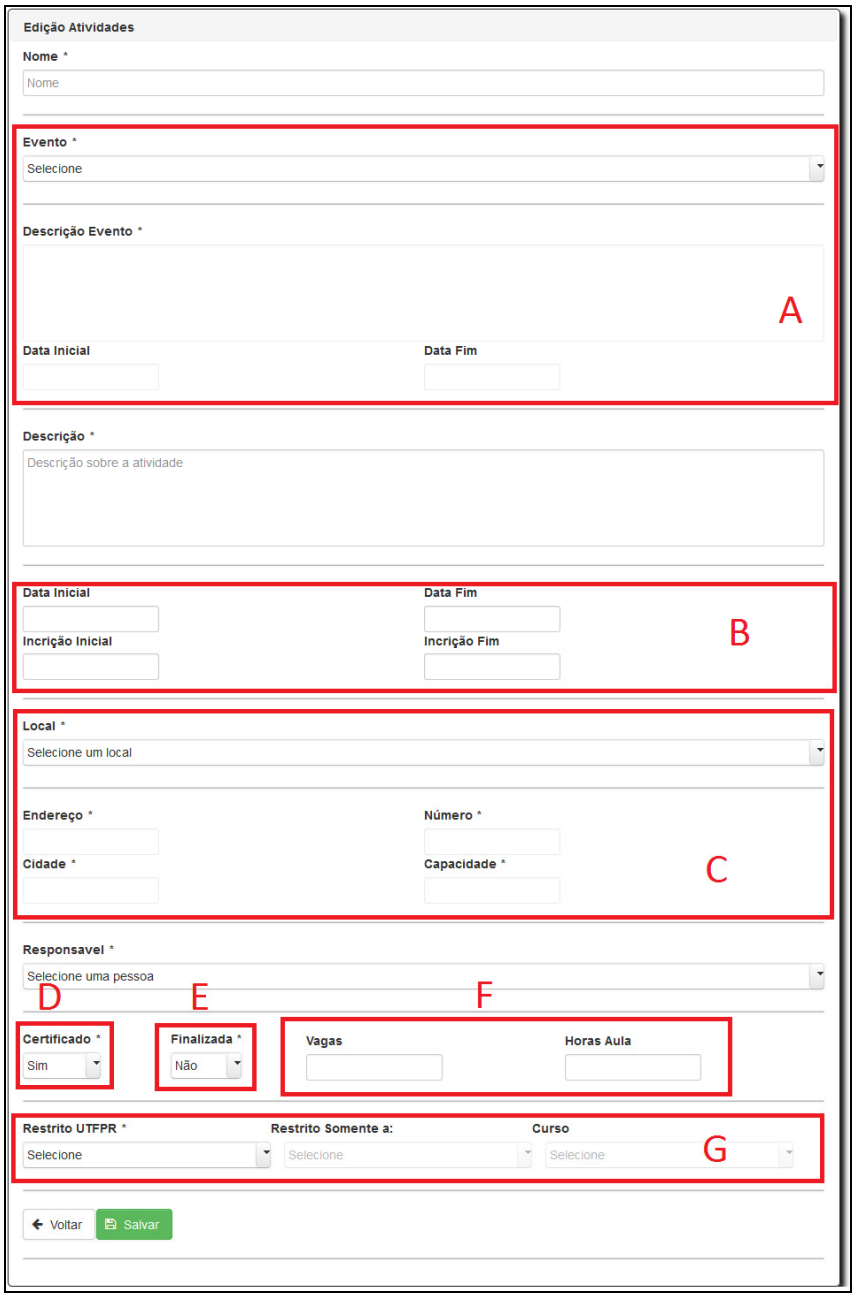

Figura 27 - Tela de Cadastro de Atividades

No formulário de atividades apresentado na Figura 27, algumas funcionalidades são destacadas:

a) Ao selecionar o evento ao qual a atividade está vinculada, os campos Descrição do Evento, Data Inicial e Data Final do evento, são automaticamente preenchidos via AJAX, continuando bloqueados para edição, somente visualizados.

b) Os campos Data Inicial e Final da atividade devem estar contidos no período de Data Inicial e Final da Atividade. E as datas inicial e final de inscrição devem representar um período anterior à realização da atividade.

c) Ao selecionar um local para realização da atividade os campos de endereço e capacidade são automaticamente preenchidos via AJAX.
d) No campo Certificado é informado se a atividade permitirá emissão de certificados, tanto pelo usuário como envio automático pelos administradores.
e) No campo Finalizada é informado se a atividade já foi realizada, permitindo o envio de certificados via e-mail, ou emissão pelo usuário.
f) Os campos Vagas e Horas aula, dizem respeito ao número de vagas da atividade, e a quantia de horas aula que a mesma terá validade, sendo esta informação também impressa no certificado.
g) A opção Restrito UTFPR, marca se a atividade será restrita somente a usuários vinculados a UTFPR. Quando marcada como sim, os campos "Restrito Somente a" e "Curso" são habilitados via AJAX. Tais campos representam a restrição a Alunos ou Servidores e a qual curso ou departamento estes terão acesso. Se não selecionada opção alguma, a atividade então é considerada como livre a todos.

Ao salvar o cadastro, uma mensagem de sucesso ou erro é apresentada ao usuário, como padrão em outras telas. A edição do cadastro foi implementada na forma de um botão no *grid*, que tem a função de redirecionar o usuário a mesma página de cadastro, trazendo desta vez os campos preenchidos com os dados do evento selecionado. Por meio da tela de edição também é possível efetuar a exclusão do cadastro, com a restrição de que não podem existir inscrições efetuadas na atividade selecionada. A Figura 28 exibe o grid de Atividades.

Figura 28 - Grid com as Opções de Editar e Certificados no *grid* de Atividades

Na Figura 28 destaca-se também a opção Certificados. A qual somente é habilitada quando uma atividade é marcada como finalizada e tem por função o envio de certificados automáticos por e-mail. Os certificados são emitidos e enviados como anexos em formato PDF, automaticamente, para os e-mails vinculados aos usuários que tiveram presença confirmada no evento. Também é possível que o próprio usuário faça a emissão de seu certificado, utilizando o menu exibido na Figura 29.

Figura 29 - Tela Minhas Atividades

Acessando o menu em destaque, o usuário é levado para uma tela que possui um *grid* contendo todas as atividades nas quais o mesmo já se inscreveu, status de presença, e a opção para emitir seu certificado. Essa opção somente é habilitada caso o usuário tenha presença e a atividade já tenha sido finalizada.

A Figura 30 apresenta o relatório que tanto pode ser emitido pelo usuário, como enviado via e-mail automaticamente, contendo os dados da atividade, bem como a assinatura vinculada ao responsável pela mesma e a carga horaria informada no cadastro da atividade.


UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Republica Federativa do Brasil
Ministério da Educação
Universidade Tecnológica Federal do Paraná

Certificamos Diogo Satdl Participou da atividade Cerimônia de Abertura promovida pela Universidade Tecnológica Federal do Paraná, no periodo de 06/09/15 18:00 a 06/09/15 23:00 com carga horária de 1 horas aula.

Pato Branco, Segunda-feira 26 Outubro 2015

Palestrante Um

Figura 30 - Exemplo de Certificado emitido pelo Sistema

4.3.5 O PROCESSO DE INSCRIÇÃO

Um dos requisitos mais importantes levantados para o sistema, diz respeito ao controle de inscrições em atividades. Este requisito foi implementado de maneira que todos os usuários do sistema tenham acesso aos menus Eventos e Atividades Abertas. O menu de eventos é mostrado na Figura 31.

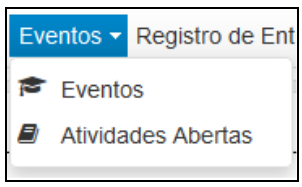

Figura 31 - Menu Eventos

Por meio da opção Atividades Abertas, o usuário pode visualizar atividades em aberto as quais o mesmo possui acesso a se inscrever. A opção Eventos redireciona o usuário a uma página contendo a listagem de todos os eventos que possuem atividades em aberto e que o usuário possui acesso a se inscrever. Na tela

de eventos é exibido o botão Programação, que envia o usuário para uma página listando as atividades para este evento, sendo esta, a mesma que pode ser acessada por usuários administradores por meio do cadastro do evento, somente acessada por uma URL diferente. A Figura 32 apresenta as informações apresentadas na página Eventos.

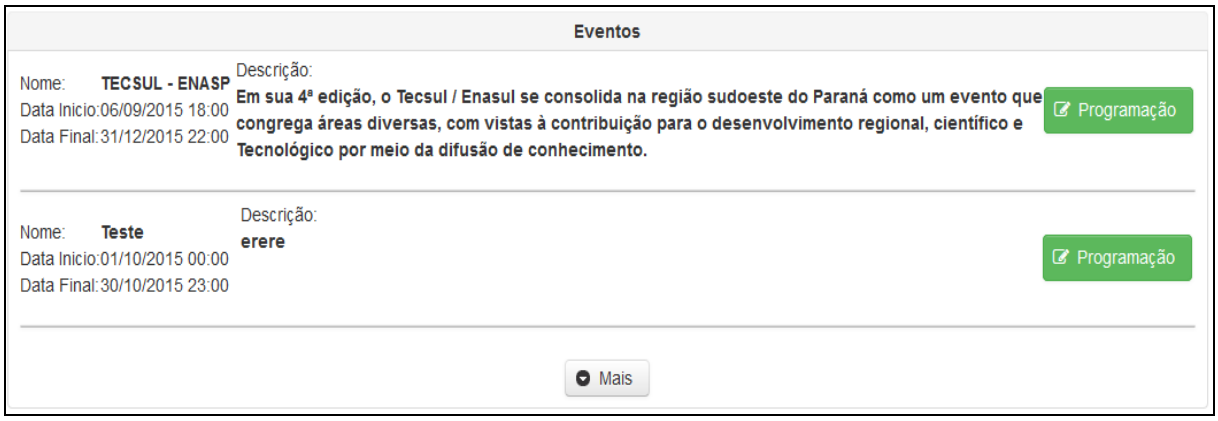

Figura 32 - Tela Eventos

As páginas Atividades em Aberto e Programação possuem o mesmo leiaute, somente sendo alterado os dados carregados. A tela Atividades em Aberto traz a listagem de todas as atividades em aberto que o usuário possa se inscrever, e a página programação, traz somente as atividades vinculadas ao evento previamente selecionado. A Figura 33 demonstra a página de Atividades Abertas.

Figura 33 - Tela Atividades em Aberto

Na tela de Atividades em Aberto o botão Inscrever-se é ocultado e em seu lugar é exibido um rótulo com a mensagem Inscrições Encerradas, quando o período

de inscrições é expirado. Ao selecionar uma atividade e clicar no botão para efetuar a inscrição o usuário é então redirecionado para a página de inscrições. Nessa página são informados os dados da atividade e opções para confirmar a inscrição ou voltar. Ao confirmar a inscrição é apresentada uma mensagem de sucesso ao usuário e o botão fica bloqueado, para evitar que o mesmo se inscreva mais de uma vez na mesma atividade. Ao retornar a listagem de atividades, é ocultada a opção inscrever-se, e em seu lugar é apresentado um rótulo contendo a mensagem “Inscrito” e número de vagas disponíveis é atualizado. A Figura 34 apresenta a página de inscrição na atividade.

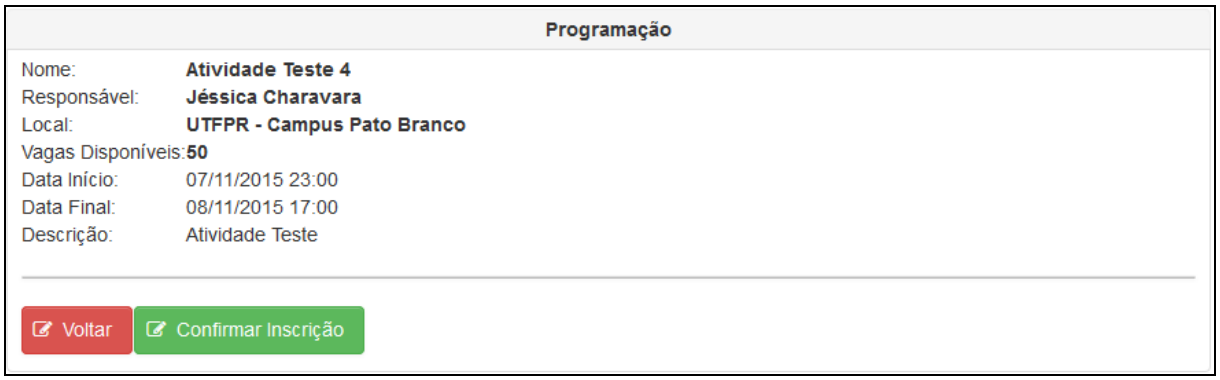

Figura 34 - Tela de Inscrição

4.3.6 REGISTRO DE ENTRADAS

A necessidade da implementação de um controle de entradas foi apresentada logo nas primeiras fases do levantamento de requisitos. Este controle deveria contar com a possibilidade de leitura de código de barras do crachá do aluno via leitor. O controle foi desenvolvido em forma de uma página especialmente projetada para essa atividade.

O Registro de Entradas possui dois campos: Atividade, em que são listadas as atividades em aberto para seleção de qual deseja-se fazer o registro; e o campo RA ou CPF que é habilitado via AJAX assim que uma atividade é selecionada, essa tela pode ser visualizada na Figura 35.

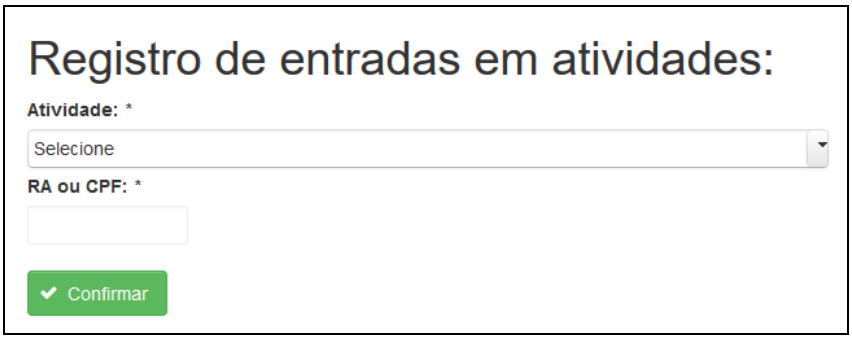

Figura 35 - Tela de Registro de Entradas

Assim que o campo Registro Acadêmico ou CPF fica disponível, é possível a digitação ou leitura do código de barras do crachá do aluno. Caso o mesmo seja acadêmico da UTFPR, parte-se do pressuposto que o aluno possua em mãos o crachá emitido pela instituição, que contém o número do Registro acadêmico. Não sendo ele aluno da UTFPR, pode-se então fazer a utilização de um de um crachá emitido por uma rotina especificamente desenvolvida dentro do sistema. Este crachá, visto que o aluno não é acadêmico da UTFPR, não possui um Registro Acadêmico válido, é impresso com seu CPF em forma de código de barras seguindo o padrão CODABAR, contendo a máscara padrão (XXX.XXX.XXX-XX). As Figuras 36 e 37 apresentam o menu de acesso à emissão do crachá e um exemplo de crachá emitido pelo sistema.

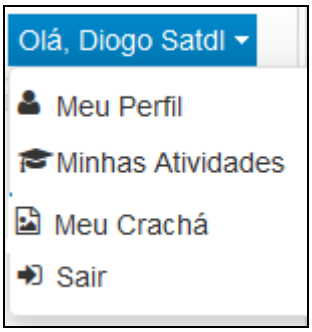

Figura 36 - Menu de acesso Meu Crachá

Figura 37 - Exemplo de Crachá emitido pelo Sistema

O crachá exibido na Figura 37 contém os dados do usuário, bem como sua foto, que pode ser alterada acessando o menu Meu Perfil, o qual contém os dados do usuário que podem ser alterados: Nome, e-mail, telefone, senha e foto. Os campos restantes não podem ser alterados pelo próprio usuário por razões de controle de acesso e por conterem ligações com outras tabelas do sistema. A Figura 38 exibe a página para manutenção de perfil do usuário.

Figura 38 - Tela de Manutenção de Usuário

Leitores de código de barra por padrão, ao final da leitura inserem o caractere *Enter*, similar a efetuar o processo de digitação manual e pressionar *Enter* ao final. A rotina de registro de entradas foi desenvolvida implementando um controle que ao pressionar a tecla *Enter* no campo RA ou CPF, o sistema faz a busca em banco de dados pelo usuário, usando como parâmetro o valor informado de CPF ou Registro Acadêmico. Ao encontrar o usuário no banco de dados, o sistema faz a busca nas inscrições, usando como parâmetro o usuário informado e a atividade selecionada. Estando o usuário inscrito na atividade, é atualizado o campo presença do registro de inscrição para “S”, marcando assim a presença do mesmo na atividade. Ao final desse processo é exibida uma mensagem de sucesso ou erro, caso o usuário não esteja cadastrado ou inscrito na atividade.

Posterior à finalização da atividade, é disponibilizado o relatório Lista de Presença. O mesmo possui como filtro a atividade a qual se quer verificar a lista de presença. Esse relatório pode ser acessado por usuários administradores por opção disponibilizada à direita do menu Registro de entradas. As Figuras 39 e 40 apresentam a tela de filtro para o relatório e um exemplo de Lista de Presença, respectivamente.

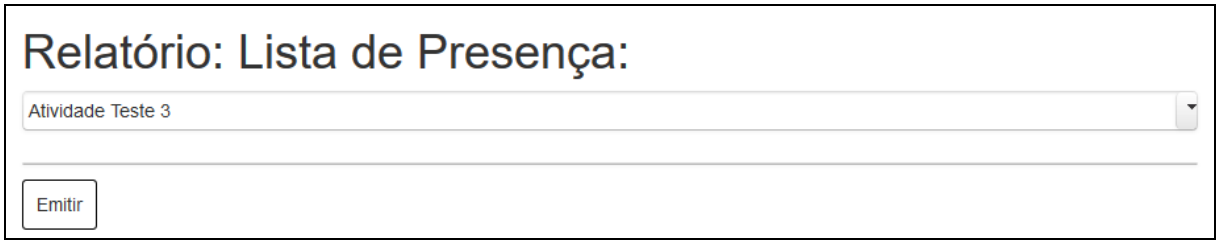

Figura 39 - Tela de Filtro para Lista de Presença

Figura 40 - Exemplo de Lista de Presença

4.4 IMPLEMENTAÇÃO DO SISTEMA

Nesse capítulo serão mostradas algumas listagens de códigos que compõe o sistema.

4.4.1 MAPEAMENTO DE ENTIDADES E CONFIGURAÇÕES

Durante o desenvolvimento da aplicação proposta foi possível utilizar algumas características importantes de padrão de desenvolvimento no ambiente Java EE. Uma das mais importantes é o mapeamento de entidades relacionais de classes do sistema, representando as tabelas do banco de dados. A Listagem 1 apresenta o exemplo de mapeamento da classe Usuário.

```
@Entity
@Table(name = "universidade")
public class Universidade implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer idUniversidade;

    @Column(name = "universidade", length = 250, nullable = false)
    private String universidade;

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "fk_endereco", referencedColumnName = "idEndereco",
nullable = false)
    private Endereco endereco;

    public Universidade() {
    }

    public Universidade(String universidade, Endereco endereco) {
        this.universidade = universidade;
        this.endereco = endereco;
    }
```

Listagem 1 - Visibilidade Menu Lateral

Na Listagem 1 destacam-se algumas características:

a) A anotação *@Entity* denota que esta classe representa uma entidade, mapeando a mesma como uma tabela do banco de dados. Sempre que a aplicação for inicializada no servidor, as tabelas do banco de dados serão

criadas ou atualizadas (no caso de já existirem), de acordo com as classes que contém essa anotação.

b) A anotação *@Table* está relacionada ao nome da tabela que será criada ou buscada para atualização no banco de dados.

c) As anotações *@Id* denota que este atributo, representa a chave primária da tabela e a anotação seguinte, *@GeneratedValue(strategy = GereratedType.AUTO)* faz que tal campo seja gerado automaticamente, sendo então auto incremento.

d) O restante dos atributos é anotado com a anotação *@Colunm* e entre parênteses propriedades específicas da tabela, como seu nome (*name*), tamanho (*length*), se permite nulo (*nullable*). Essas colunas são criadas com o tipo padrão do mapeamento, exemplo: Atributo do tipo *String* será gerado como *VARCHAR*, *int* como *INTEGER*, Double como *NUMBER* ou *DECIMAL* dependendo do banco de dados. Nada impede que o tipo seja definido manualmente por meio da definição *collunmDefinition*.

e) No caso de uma tabela possuir uma chave estrangeira, conforme o exemplo da listagem, é necessário o mapeamento com as anotações *@OneToOne* para representar ligação de um para um. Também é possível usar anotações como *@OneToMany* ou *@ManyToOne* conforme a necessidade. Em sequência a tal anotação deve representar a coluna e tabela de ligação através da anotação *@JoinColunm*. Tal anotação possui as propriedades obrigatórias: *name* sendo o nome da chave estrangeira que será gerada, *referencedColunmName* que representa a chave primária da tabela a ser ligada. Em seguida deve seguir o atributo representando a classe de entidade ligada a esta. No caso do exemplo, foi ligada a classe Universidade a classe Endereço, que por sua vez é outra classe de entidade, também representando uma tabela do banco de dados.

Esse mapeamento foi utilizado como padrão para todas as classes do sistema, representando todas as tabelas do banco de dados da aplicação. Por meio da utilização desse tipo de mapeamento, é possível não somente eliminar o trabalho de criar o banco de dados manualmente, mas também, evitar erros de mapeamento de entidade manual.

Sendo uma vez mapeadas as classes, é necessária a configuração do arquivo de propriedades do *Spring Boot*, exibido na Listagem 2. Este arquivo tem por

principal função eliminar a configuração via XML, antigamente necessária para que o mapeamento relacional do *framework* Spring Data fosse inicializado.

```
spring.datasource.url: jdbc:mysql://localhost/tcc
spring.datasource.username=root
spring.datasource.password=12345
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.hbm2ddl.auto=update
spring.jpa.show-sql: true
spring.jpa.properties.hibernate.format_sql=true
```

Listagem 2 - Arquivo de Configuração do Spring Boot.

As configurações exibidas na Listagem 2 representam as mesmas que anteriormente necessitariam ser configuradas via um arquivo especifico de configurações do *Hibernate*, que por sua vez está presente no *framework Spring Data*. Nessas configurações são informadas a URL do banco de dados, o usuário e senha, nome do driver de conexão, *update* ou *create* nas tabelas do banco de dados, caso estejam diferentes das classes mapeadas ao rodar a aplicação e mostrar SQL formatado no console da IDE, respectivamente.

O restante das configurações via arquivos de XML não se fazem mais necessárias utilizando o *Spring Boot*, que fica responsável por toda a inicialização do *framework* via uma classe especialmente gerada para tal função.

4.4.2 REPOSITÓRIOS E INJEÇÃO DE DEPENDÊNCIAS

A utilização do *framework Hibernate* exige por padrão, que sejam criados *Data Object Access* (DAO) para acesso a dados de duas formas: a criação de um dao genérico, contendo as quatro operações *Create, Read, Update and Delete* (CRUD) padrão e a criação de dados específicos para cada classe, contendo seus métodos particulares. Ou a criação de um DAO para cada classe. Visando minimizar tal trabalho, o *framework Spring Data* conta com a interface *CrudRepository*, que já contém implementado as operações padrão em banco de dados. A Listagem 3 exemplifica o uso de um repositório para a classe curso.

```
public interface CursoRepository extends CrudRepository<Curso, Integer>{

    List<Curso> findByUniversidade(Universidade un);

    List<Curso> findByTipoAndUniversidade(String tipo, Universidade u);

}
```

Listagem 3 - Exemplo de Repositório.

Na Listagem 3 é possível observar o funcionamento do repositório. É criada uma interface que estende a classe *CrudRepository*, a qual recebe como parâmetro a classe da aplicação a qual deseja-se criar o repositório e o tipo de dados que representa sua chave primária no banco de dados. Apenas o código exibido já é o suficiente para que se tenha disponíveis os métodos *save* que faz inserção ou atualização no banco dados, *delete* responsável por efetuar operações de exclusão, *findAll* e *findById* que fazem operações de *select* trazendo todos os registros, ou buscando somente um pela chave primaria, respectivamente.

Além dos métodos padrão, é possível implementar métodos específicos de busca, como no caso da listagem. Para isso, é necessário que o método seja nomeado como *findBy* seguido da propriedade especifica classe. Como no exemplo da listagem, o método *findByUniversidade*, recebe como parâmetro uma instância da classe Universidade, efetua a busca no banco de dados e retorna uma lista de Cursos que estão ligados a uma determinada Universidade recebida por parâmetro.

Dada a necessidade de utilizar mais de um parâmetro para efetuar a busca, basta que o nome do método seja conectado com a próxima propriedade, como observado no exemplo o método *findByTipoAndUniversidade*, que recebe como parâmetro uma *string* contento o tipo da universidade e a Universidade em si. Este tipo de implementação é possível graças ao *framework Spring Data*, que faz a leitura do nome do método, relacionando com as propriedades da classe passada como parâmetro ao estender o repositório, montando automaticamente a instrução SQL de busca no banco de dados. A utilização de repositórios foi adotada como padrão para a aplicação, contendo um repositório para cada classe.

O sistema proposto foi desenvolvido seguindo o padrão MVC. Sendo assim, as classes de mapeamento ficam responsáveis pela parte de modelo, as páginas JSF desenvolvidas responsáveis pela visualização, e os *beans* e *daos* pelo controle.

Os *beans*, que são objetos diretamente ligados às páginas JSF possuem algumas particularidades. Na Listagem 4 destacam-se algumas anotações:

a) A anotação @*Controller* tem a responsabilidade de informar ao *Spring Framework* que essa classe representa um *bean* de controle.
b) @*ManagedBean* repassa a página JSF o nome do *bean* a ser acessado diretamente.
c) @*RequestScoped* tem a responsabilidade de representar esta classe, como uma classe de *Request*, ou seja, tal objeto será criado somente quando requisitado pela página. O JSF possui outros escopos como o *SessionScoped* para objetos que se manterão instanciados durante toda a seção por exemplo.
d) A anotação @*Autowired* injeta a dependência dos repositórios necessários para os métodos desta classe. Dessa maneira sempre que a classe CursoBean for instanciada, também serão instanciados os repositórios anotados com a injeção. Com isso evita-se a necessidade de instanciar manualmente os repositórios, evitando erros geralmente associados a instancias de objetos nulos.
e) Os métodos seguintes fazem uso dos métodos do repositório para efetuar operações em bancos de dados. Por exemplo, o método *carregar* é chamado sempre que a página é acessada, e faz com que as propriedades *lstCursos* e *lstUniversidades* sejam carregadas com listas contendo todos os registros do banco de dados por meio do método *findAll* do repositório. Essas listas são utilizadas para popular os *grids* das páginas do sistema. Os métodos seguintes seguem o mesmo padrão, efetuando diferentes operações no banco de dados.
f) O método *novo* é chamado na página, quando o usuário clicar no botão de Cadastro. O mesmo instancia um novo objeto do tipo Curso, que é carregado com os dados informados na página. Ao realizar operações no banco de dados esse objeto é utilizado. Este objeto, também é utilizado quando o usuário seleciona um registro no *grid* do sistema. Uma vez selecionado, o objeto *cursoCadastro* recebe o objeto selecionado no *grid*. Então, pode-se efetuar as validações *verificaEditar* e *verificaExcluir*, que tem a responsabilidade de conferir se o usuário clicou em algum item no

grid, ou seja, se o *cursoCadastro* não é nulo, e assim exibir a caixa de diálogo com seus dados.

```
@Controller
@ManagedBean(name = "cursoBean")
@RequestScoped
public class CursoBean implements Serializable {

    @Autowired
    CursoRepository cursoRepository;
    @Autowired
    UniversidadeRepository universidadeRepository;
    private Curso cursoCadastro;
    private Universidade universidade;
    private List<Curso> lstCursos;
    private List<Curso> lstCursoUniversidade;
    private List<Curso> lstCursosFiltrados;
    private List<Universidade> lstUniversidades;
    private int count;

    @PostConstruct
    public void init() {
        cursoCadastro = new Curso();
    }
    public void novo() {
        cursoCadastro = new Curso();
    }
    public void carregar() {
        lstCursos = (List<Curso>)cursoRepository.findAll();
        lstUniversidades = (List<Universidade>)universidadeRepository.findAll();
    }
    public void salvar(){
        RequestContext context = RequestContext.getCurrentInstance();
        try{
            cursoRepository.save(cursoCadastro);
            FacesUtil.addMsgInfo("Cadastro salvo com sucesso!");
            novo();
            context.execute("PF('dlgNovo').hide();");
        }catch (Exception e){
            FacesUtil.addMsgError("Erro ao gravar o cadastro. Erro: " + e.getMessage());
        }
    }
    public void verificaEditar(){
        RequestContext context = RequestContext.getCurrentInstance();
        if(cursoCadastro == null){
            FacesUtil.addMsgError("Selecione um cadastro!");
        }else{
            context.execute("PF('dlgEditar').show();");
        }
    }
    public void editar(){
        RequestContext context = RequestContext.getCurrentInstance();
        try{
            cursoRepository.save(cursoCadastro);
            FacesUtil.addMsgInfo("Cadastro editado com sucesso!");
            novo();
            context.execute("PF('dlgEditar').hide();");
        }catch (Exception e){
            FacesUtil.addMsgError("Erro ao gravar o cadastro. Erro: " + e.getMessage());
        }
    }
```

Listagem 4 - Exemplo de Bean do Sistema

4.4.3 BLOQUEIO DE ACESSO A PÁGINAS RESTRITAS

Durante o levantamento de requisitos, foi apresentada a necessidade de bloquear o acesso às páginas de cadastro a usuários sem direto de administrador. Esse requisito foi implementado usando a interface padrão *PhaseListener* do JSF. Para utilização dessa implementação, é necessária a criação de uma classe que implementa os métodos dessa interface. A classe implementada será chamada ao ser realizado o acesso a cada página do sistema, fazendo a interceptação do acesso. A mesma deve implementar obrigatoriamente os seguintes métodos:

a) *afterPhase*: Código que será executado após o acesso a página.
b) *beforePhase*: Código será executado antes da página ser renderizada.
c) *getPhaseId*: Retorna o código de acesso a página, gerado automaticamente para cada acesso a cada página.

A Listagem 5 exibe a implementação desenvolvida para a aplicação.

```
public class LoginPhaseListener implements PhaseListener {
    @Override
    public void afterPhase(PhaseEvent phaseEvent) {
    }
    @Override
    public void beforePhase(PhaseEvent phaseEvent) {
        ELContext elContext = FacesContext.getCurrentInstance().getELContext();
        LoginBean bean = (LoginBean)   FacesContext.getCurrentInstance().getApplication()
                .getELResolver().getValue(elContext, null, "loginBean");
        FacesContext context = phaseEvent.getFacesContext();
        UIViewRoot uiViewRoot = context.getViewRoot();
        String pagina = uiViewRoot.getViewId();
        boolean restrita = pagina.contains("restrict");
        if(restrita){
            if(bean.getUsuarioLogado().getNome() == null){
                Application app = context.getApplication();
                NavigationHandler nav = app.getNavigationHandler();
                nav.handleNavigation(context,null, "/pages/public/login.xhtml?faces-redirect=true");
            }else if(pagina.contains("usuarios")
                    && bean.getUsuarioLogado().getRole().equalsIgnoreCase("USER")){
                Application app = context.getApplication();
                NavigationHandler nav = app.getNavigationHandler();
                nav.handleNavigation(context,null, "/pages/restrict/index.xhtml?faces-redirect=true");
            }else if(pagina.contains("cidades")
                    && bean.getUsuarioLogado().getRole().equalsIgnoreCase("USER")){
                Application app = context.getApplication();
                NavigationHandler nav = app.getNavigationHandler();
                nav.handleNavigation(context,null, "/pages/restrict/index.xhtml?faces-redirect=true");
            }else if //código omitido, contém o bloqueio a todas as outras
                                //páginas de cadastro do sistema.
        }
}
```

Listagem 5 - Classe PhaseListener

Na Listagem 5 é possível observar que o método *afterPhase* não é utilizado, pois a necessidade é de bloquear a página antes do acesso, logo, foi implementado o método *beforePhase*. No mesmo é recuperado o contexto do acesso e por meio

dele é possível acessar o *bean* de *login* do usuário, uma vez que o mesmo é um *bean* de seção e se mantém na memória durante todo o acesso do usuário. Por meio desse objeto é possível fazer o redirecionamento para a página de *login*, caso o usuário não esteja logado, ou, redirecioná-lo a página principal caso tente acessar diretamente pela URL páginas restritas.

Estando uma vez logado no sistema os menus e botões aos quais o usuário não tem acesso, não são renderizados usando a propriedade *rendered*, presente nos componentes do *framework Primefaces*. A Listagem 6 apresenta um exemplo desse bloqueio.

```
<p:menuitem value="  Cidades"
outcome="/pages/restrict/cidade/cidades.xhtml"
            icon="fa fa-globe"
            rendered="#{loginBean.usuarioLogado.role == 'ADM'}"/>
<p:menuitem value="  Universidades   "
outcome="/pages/restrict/universidade/universidades.xhtml"
            icon="fa fa-university"
            rendered="#{loginBean.usuarioLogado.role == 'ADM'}"/>
<p:menuitem value="  Cursos/Deptos   "
outcome="/pages/restrict/curso/cursos.xhtml"
            icon="fa fa-building"
            rendered="#{loginBean.usuarioLogado.role == 'ADM'}"/>
```

Listagem 6 - Exemplo de exibição Dinâmica de Componentes

Os componentes representados na listagem serão apresentados caso o usuário utilizado possua como propriedade o campo role igual a ADM. Não sendo atingida tal condição, tais menus não são apresentados ao usuário, fazendo assim, um modelo dinâmico de apresentação, de acordo com o usuário utilizado.

4.4.4 ENVIO DE CERTIFICADOS VIA E-EMAIL

Um dos métodos mais importantes do sistema tem a responsabilidade de efetuar o envio dos certificados via e-mail. Esse método recebe por parâmetro uma lista de objetos *usuarioAtividade*, que contém os usuários matriculados em uma atividade previamente selecionada, conforme pode ser visualizado na Listagem 7. Ao efetuar a chamada deste método, é percorrida a lista recebida, verificando quais usuários tiveram presença confirmada na atividade. Ao atingir tal condição é então criado um certificado em formato *Portable Document Format* (PDF) na máquina servidor, e anexado no *e-mail*. Em seguida é acessada a propriedade e-mail do

usuário, e enviado o e-mail. O método repete este ciclo sobrescrevendo o arquivo local para cada usuário da lista.

```
public Integer sendCertificados(List<UsuarioAtividade> usuarioAtividadeList) {
    Properties props = new Properties();
    /** Parâmetros de conexão com servidor Hotmail */
    props.put("mail.transport.protocol", "smtp");
    props.put("mail.smtp.host", "smtp.live.com");
    props.put("mail.smtp.socketFactory.port", "587");
    props.put("mail.smtp.socketFactory.fallback", "false");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.port", "587");

    javax.mail.Session session = javax.mail.Session.getInstance(props,
            new javax.mail.Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("email", "senha");
                }
            });
    session.setDebug(true);
    try {
        for (int i = 0; i < usuarioAtividadeList.size(); i++){
         if(usuarioAtividadeList.get(i).getPresenca().equals("S")){
                emitir(usuarioAtividadeList.get(i).getAtividade(),
usuarioAtividadeList.get(i).getUsuario());
                Message message = new MimeMessage(session);
                message.setFrom(new InternetAddress("eventos@utfpr.edu.br"));
//Remetente
                BodyPart messageBodyPart = new MimeBodyPart();
                messageBodyPart.setText("Segue em anexo seu certificado!");

                Multipart multipart = new MimeMultipart();
                multipart.addBodyPart(messageBodyPart);
                messageBodyPart = new MimeBodyPart();
                DataSource source = new
FileDataSource("C:\\Programacao\\Projetos\\Java\\TCCJSF\\certificado.pdf");
                messageBodyPart.setDataHandler(new DataHandler(source));
                messageBodyPart.setFileName("Certificado.pdf");
                multipart.addBodyPart(messageBodyPart);
                message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(usuarioAtividadeList.get(i).getUsuario().getEmail()));
//Destinatário(s)
                message.setSubject("Certificado UTFPR");//Assunto
                message.setText("Segue em anexo o seu certificado!");
                message.setContent(multipart);
                Transport.send(message);
            }
        }
        return 1;
    } catch (MessagingException e) {
        throw new RuntimeException(e);
```

Listagem 7 - Classe de Controle de Acesso as Páginas

4.4.5 BLOQUEIO DE ACESSO A ATIVIDADES

As atividades do sistema proposto são listadas de acordo com o curso, universidade e tipo do usuário que está efetuando o acesso. Esse controle se fez necessário para evitar que usuários se inscrevam em atividades que não dizem respeito ao seu curso ou universidade, ou até mesmo atividades restritas a servidores.

As páginas de listagem de dados do sistema estão ligadas a classes que efetuam buscas em bancos de dados e montam listas que são repassadas novamente as páginas, e então renderizadas em forma de *grid*. No contexto das atividades, a aplicação contém duas páginas que possuem tais listagens: a página de programação do evento e a página que faz a listagem completa das atividades que o usuário tem acesso.

A fim de implementar esse controle, foram desenvolvidos dois métodos que efetuam as buscas no banco de dados, usando como parâmetro o usuário logado. A Listagem 8 apresenta o método que é chamado na página de programação do evento.

```
public void carregarAtivPorEvento(Usuario u) {

    try {
        String valor = FacesUtil.getParam("idEvento");
        if (valor != null) {
            //Verifica se o parametro nao é nulo e efetua a busca
            Integer codigo = Integer.valueOf(valor);
            evento = eventoRepository.findOne(codigo);

            //Faz o mesmo processo do metodo acima.
            List<Atividade> lsuUtil1 =
atividadeRepository.findByFinalizadaAndRestritoAAndCurso("N", u.getTipo(),
u.getCurso());
            List<Atividade> lsuUtil2 =
atividadeRepository.findByFinalizadaAndRestrito("N", "N");

            //Nova lista, que receberá somente as atividades ligadas ao
evento passado por parametro.
            List<Atividade> lstUtil3 = new ArrayList<>();

            //Percorre as duas listas adicionando na terceira
            for(Atividade a : lsuUtil1){
                if(a.getEvento().getIdEvento() == evento.getIdEvento()){
                    lstUtil3.add(a);
                }
            }
            for(Atividade a : lsuUtil2){
                if(a.getEvento().getIdEvento() == evento.getIdEvento()){
                    lstUtil3.add(a);
                }
            }
```

```
            lstAtividades = lstUtil3;
        }
    } catch (RuntimeException ex) {
        FacesUtil.addMsgError("Erro ao tentar carregar. Erro: " +
ex.getMessage());
    }
}
```

Listagem 8 - O método para carregar atividades por Evento

Observando a Listagem 8, é possível verificar o funcionamento do método. Recupera-se o código do evento passado por parâmetro, logo em seguida o evento é buscado no banco de dados. Por meio desse evento e do usuário recebido, são recuperadas duas listas de atividades. A primeira apresenta as atividades abertas, que contém a mesma restrição do tipo do usuário e o mesmo curso. A segunda lista por sua vez, traz todas as atividades abertas que não possuem restrição alguma. Após a busca são percorridas as duas listas de atividades, verificando quais estão ligadas ao evento passado por parâmetro, ao ser atingida tal condição, são então adicionadas a uma terceira lista, que é retornada a página.

A página contendo todas as atividades que o usuário possui acesso, apresenta um método semelhante, com o diferencial de não efetuar a comparação por evento, e sim, por universidade do usuário, retornando todas as atividades sem restrição de evento, como pode ser observado na Listagem 9.

```
public void carregarAtivAbertas(Usuario u) {

    //Verifica a universidade do usuario
    if(u.getInstEnsino().getUniversidade().contains("UTFPR")){

        //Caso seja da UTFPR, temos 2 casos, ou a atividade é restrita a alunos/servidores de um
curso/depto
        List<Atividade> lsuUtil1 = atividadeRepository.findByFinalizadaAndRestritoAAndCurso("N",
u.getTipo(), u.getCurso());
        //ou é de livre acesso
        List<Atividade> lsuUtil2 =  atividadeRepository.findByFinalizadaAndRestrito("N", "N");
        //Adicionamos as duas listas a apenas uma lista que será utilizada
        for(Atividade a : lsuUtil2){
            lsuUtil1.add(a);
        }
        lstAtividades = lsuUtil1;

    //Caso o usuario nao seja da UTF, retorna todos as atividades em aberto e nao restritas.
    }else{
        lstAtividades = atividadeRepository.findByFinalizadaAndRestrito("N", "N");
    }

}
```

Listagem 9 - O método para Carregar Atividades Abertas

5 CONCLUSÃO

Com o desenvolvimento deste projeto, foi possível conhecer a importância de um sistema que gerencie as atividades realizadas por instituições de ensino ao longo do período letivo. Como relatado no decorrer do projeto, esse controle antes era efetuado totalmente de forma manual, sem um controle que automatizasse tarefas visando minimizar o trabalho e interferência de pessoas.

O gerenciamento de eventos envolve uma série de processos, tais como a emissão de certificados, o envio de *e-mails*, a emissão de relatórios e vários cadastros necessários para manutenção dessas atividades. O sistema desenvolvido supre a necessidade de se efetuar esses controles de maneira automatizada, eliminando os controles manuais para todos esses processos.

Ao utilizar a rotina de envio de certificados via e-mail, a instituição elimina a necessidade de impressão e entrega manual desses documentos. Também ao se fazer uso do processo de controle de entradas em atividades, é eliminada a necessidade de fichas de presença e coleta de assinaturas manuais, as quais faziam deste controle lento e arriscado, visto que esses documentos correm o risco de se perderem com o tempo.

A linguagem Java provou ser adequada para o desenvolvimento *web* e supriu todas as necessidades do projeto, permitindo que todos os requisitos levantados fossem implementados. Os *frameworks* utilizados em conjunto com a linguagem, tornaram o processo mais ágil e de fácil execução.

As ferramentas utilizadas durante o desenvolvimento do software provaram ser de extrema eficácia. O *Intellij IDEA* tornou, tanto a integração com os *frameworks* utilizados quanto com o servidor de aplicações *JBoss WildFly*, simples e rápida. A ferramenta *Visual Paradigm* permitiu que o desenvolvimento da documentação e modelagem de forma eficaz, bem como o servidor de aplicações *WildFly*, que desempenhou suas funções de forma satisfatória. Por fim o banco de dados *MySQL* foi capaz de suprir todas as necessidades do projeto.

Todos os requisitos levantados durante a fase de análise foram desenvolvidos usando como base a linguagem Java e sua implementação *Enterprise Edition*, sendo possível atingir o objetivo proposto de desenvolver um *software web* para gerenciamento de eventos.

As principais dificuldades encontradas no desenvolvimento foram, primeiramente, o fato de não possuir total domínio do *framework* Spring, exigindo certo tempo de pesquisa e busca de aprendizado tanto em conversas com o orientador quanto em documentação oficial e fóruns da internet.

Outra grande dificuldade encontrada durante o desenvolvimento, diz respeito a algumas rotinas especificas do sistema. Como por exemplo, as implementações de *upload* de imagens e envio de *e-mail*, que exigiram o uso de tutoriais disponíveis na internet para aprendizado, adaptando tais modelos às necessidades do sistema desenvolvido.

Como trabalhos futuros pretende-se a implementação de uma opção de leitura e impressão do código do aluno junto ao código da atividade por meio de QRCode, visando aumentar o controle sobre as entradas em atividades e facilitar ainda mais o processo por parte do usuário. Também é necessário o levantamento e a implementação de outros relatórios de acordo com as necessidades apresentadas.

REFERÊNCIAS

BROOKS, David R. **An introduction to HTML and JavaScript:** for Scientists and Engineers. Springler, 2007.

CALISKAN, Mert; VARAKSIN, Oleg**. Primefaces cookbook second edition**. Packt Publishing, 2013.

CMIL, Micha; MATLOKA, Michall MARCHIONI, Francesco. **Java EE 7 - development with WildFly**. Packt Publishing, 2014.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. **Sistemas distribuídos:** Conceitos e Projeto. 5ª ed. São Paulo: Bookman, 2013.

DUBOIS, Paul. **MySQL cookbook**. 3ª ed. O'Reilly Media Inc. Sebastopol, CA, USA, 2014.

GARRETT, James J. **Ajax:** A New Approach to Web Applications. 2005. Disponível em: <https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf>. Acesso em: 01 out. 2015.

GEARY, David; HORSTMANN, Cay. **Core Java Server Faces**. 3ª ed. Rio de Janeiro: Alta Books, 2012.

GONÇALVES, Antônio. **Beginning Java EE 7**. 1ª ed. Apress Berkely, CA, USA, 2013.

HIBERNATE. **Hibernate**. 2015. Disponível em: <http://www.hibernate.org>. Acesso em: 01 de outubro de 2015.

KROCHMALSKI, Jarosław. **IntelliJ IDEA essentials**. Packt Publishing, 2014.

KUROSE, James F; ROSS, Keith W. **Redes de computadores e a Internet**. 5ª ed. São Paulo: Pearson, 2011.

MIKKONEN, Tommi; TAIVALSAARI, Antero. **Web applications - spaghetti code for the 21st Century**. In: Sixth International Conference on Software Engineering Research, Management and Applications. p 319-328 IEEE Computer Society Washington, DC, USA, 2008.

MURUGESAN, Sam. **Understanding Web 2.0**. IT Professional archive, v. 9, n. 4, July 2007. p. 34-41. IEEE Educational Activities Department Piscataway, NJ, USA, 2007.

O'REILLY, Tim. **What is Web 2.0:** Design Patterns and Bussiness Models for the Next Generation of Software. 2005. Disponível em: <http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html>. Acesso em: 29 set. 2015.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. 3ª ed. Rio de Janeiro: Brasport, 2005.

SHELDON, Robert; MOES, Geoff. **Begining MySQL**. Wiley Publishing Inc. Indianapolis, IN, USA, 2005.

SPRING. **Spring Framework**. Disponível em: <http://projects.spring.io/spring-framework/>. Acesso em: 15 ago. 2015.

TANEMBAUM, Andrew S; STEEN, Martin Van. **Distributed systems:** Principles and Paradims 2nd Edition. 2ª ed. Prentice Hall Upper Saddle River, NJ, USA, 2006.

TANENBAUM, Andrew S. **Computer networks.** 4ª ed. Prentice Hall Upper Saddle River, NJ, USA, 2002.

TEAGUE, Jason C. **CSS3:** Visual QuickStart Guide. 5ª ed. Peachpit Press San Francisco, Ca, USA, 2010.

VISUAL PARADIGM. **Visual Paradigm**. Disponível em: <http://www.visual-paradigm.com/>. Acesso em: 01 out. 2015.

W3C. **What is the Document Object Model?** 2010. Disponível em: <http://www.w3.org/TR/DOM-Level-2-Core/introduction.html>. Acesso em: 01 out. 2015.

ZAKAS, Nicholas C.; MCPEAK, Jeremy; FAWCETT, Joe. **Professional AJAX**. 2ª ed. Wiley Publishing Inc. Indianapolis, IN, USA, 2007.