INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

ERIC MARINS CECCAGNO

Sistema Web para Gerenciamento dos Encontros de Orientação do Trabalho de Conclusão de Curso

PASSOFUNDO
2018

ERIC MARINS CECCAGNO

Sistema Web para Gerenciamento dos Encontros de Orientação do Trabalho de Conclusão de Curso

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Campus Passo Fundo, como requisito parcial para a aprovação na disciplina de Projeto de Conclusão II (PC II).

Orientador (a): Jorge Luis Boeira Bavaresco

PASSO FUNDO

2018

ERIC MARINS CECCAGNO

Sistema Web para Gerenciamento dos Encontros de Orientação do Trabalho de Conclusão de Curso

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Banca Examinadora:

_____

Jorge Luis Boeira Bavaresco

_____

Vanessa Lago Machado

_____

Josué Toebe

_____

Rafael Marisco Bertei

PASSO FUNDO

2018

AGRADECIMENTOS

Primeiramente agradeço a minha família, pelo carinho, apoio e sempre me incentivando a estudar e correr atrás do meu futuro.

Agradeço ao professor Jorge Luis Boeira Bavaresco que através de seus conhecimentos e experiência contribuíram grandemente para a realização desse projeto.

Agradeço aos meus colegas pela amizade e troca de conhecimentos que contribuíram com o nosso crescimento e aprendizado ao longo do curso.

Por fim, agradeço aos demais professores campus pelo conhecimento passado e aos funcionários do campus pela consideração.

RESUMO

O curso de Tecnologia de Sistemas para Internet do Instituto Federal Sul-Rio-Grandense Campus Passo Fundo requer, para obtenção do diploma, um Trabalho de Conclusão de curso. Para que se possa ter a oportunidade de apresentar o trabalho perante uma banca, pede-se o mínimo de dez encontros de orientação entre aluno orientado e professor orientador. Atualmente a documentação sobre essas orientações é feito manualmente e, geralmente, ficam a cargo do professor orientador, o que torna tal documentação informal e pode proporcionar falha na comunicação e acompanhamento entre o aluno e professor. O sistema proposto visa facilitar o gerenciamento das orientações, tornando a orientação mais formal, além do acompanhamento e comunicação de docentes e discentes aprimorado, por meio de um rápido e fácil acesso as informações que foram discutidas nas orientações prévias. O trabalho foi desenvolvido em linguagem de programação Java utilizando a plataforma *Java EE*, em conjunto com as tecnologias *JavaServer Faces*, *Java Persistence API*, *Hibernate*, *Enterprise JavaBean, Java Authentication and Authorization Service* e *JasperReports*. A modelagem do sistema foi feita com diagramas *UML*, e o sistema foi desenvolvido usando o padrão *Model-View-Controller* com a persistência dos dados feita em banco de dados *PostgreSQL*. O resultado obtido foi um sistema onde as ações são controladas através das permissões dos usuários – controlados pelo JAAS -, assim o sistema difere entre usuário Alunos, Professores e Administradores, e controla as ações disponíveis apenas para usuários autorizados para ela, variando entre o cadastro de novos usuários, edição das informações do usuário autenticado, edição e criação de fichas, encontros, documentos e a geração de relatório.

Palavras-chave: Sistema web. Encontros de orientação do TCC. Java. Java EE. JSF. JAAS. JasperReports.

ABSTRACT

The Internet Systems Technology course of the Instituto Federal Sul-Rio-Grandense Campus Passo Fundo requires, to obtain the diploma, a Course Completion Work. In order to have the opportunity to present the work before a bank, it is requested the minimum of ten orientations meetings between oriented student and guiding teacher. Currently the documentation on these guidelines is done manually and are usually handled by the tutor, which makes such documentation informal and can provide communication and monitoring failure between the student and the teacher. The proposed system aims to facilitate the management of the guidelines, making the orientation more formal, besides the monitoring and communication of teachers and students improved, through a quick and easy access to the information that were discussed in the previous guidelines. The system was developed in Java programming language using the Java EE platform, in conjunction with the technologies JavaServer Faces, Java Persistence API, Hibernate, Enterprise JavaBean, Java Authentication and Authorization Service and JasperReports. The system modeling was done with UML diagrams, and the system was developed using the Model-View-Controller standard with the data persistence done in PostgreSQL database. The result obtained was a system where the actions are controlled through the permissions of the users - controlled by JAAS -, so the system differs between users Students, Teachers and Administrators, and controls the actions available only to users authorized to it, diversifying these actions between registration of new users, editing of authenticated user information, editing and creation of records, meetings, documents and report generation.

Key words: Web System. Orientations meetings. Java. Java EE. JSF. JAAS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E DE SIGLAS

API – Application Programming Interface
CRUD – Create, Read, Update and Delete
DAO – Data Acess Object
EJB – Enterprise JavaBeans
FACNET – Faculdade de Negócios e Tecnologias da Informação
GUI – Graphical User Interface
HTML – HyperText Markup Language
IDE – Integrated Development Environment
IFC – Instituto Federal Catarinense
IFSUL – Instituto Federal Sul-rio-grandense
JAAS – Java Authentication and Authorization Service
JAVA EE – Java Enterprise Edition
JAVA ME – Java Micro Edition
JAVA SE – Java Standard Edition
JNDI – Java Naming and Directory Interface
JPA – Java Persistence API
JSF – JavaServer Faces
JTA – Java Transaction API
MVC – Model-View-Controller
ORM - Object-relational Mapping
PDF – Portable Document Format
SGBD – Sistema de Gerenciamento de Banco de Dados
TCC – Trabalho de Conclusão de Curso
TSPI – Tecnologia de Sistemas para Internet
UI – User Interface
UML – Unified Modeling Language
UNIPLAC – Universidade do Planalto Catarinense

SUMÁRIO

1. INTRODUÇÃO

O curso Superior Tecnologia de Sistemas para Internet (TSPI) do Instituto Federal Sul-Rio-Grandense (IFSUL) campus Passo Fundo requer, para obtenção do diploma, um Trabalho de Conclusão de Curso (TCC), no qual o aluno é orientado por um professor. Durante a execução do trabalho, pede-se que o aluno orientado e professor orientador devem ter, no mínimo, dez encontros de orientação, os quais são registrados em uma ficha de encontros, que além de conter os assuntos tratados, data e horário são obrigatórios para que o aluno possa apresentar seu trabalho perante a banca.

Atualmente a documentação sobre essas orientações é feito manualmente e, geralmente, ficam a cargo do professor orientador, o que torna tal documentação informal e pode proporcionar falha na comunicação e acompanhamento entre o aluno e professor.

Assim se chegou à seguinte pergunta: Como realizar o gerenciamento dos encontros de orientação de maneira mais eficiente e com rápido acesso as informações necessárias? Para resolver esse problema desenvolveu-se um sistema web que visa maior eficiência no desenvolvimento das orientações do TCC, tornando a orientação mais formal, além do acompanhamento e comunicação de docentes e discentes melhorar por meio de um rápido e fácil acesso as informações que foram discutidas nas orientações prévias.

1.1. OBJETIVOS

Com o desenvolvimento de um sistema de gerenciamento de encontros de orientação, professores e alunos poderão acessar de qualquer lugar as informações dos encontros. O sistema auxiliará com informações de data e assuntos tratados por orientação, bem como a geração de um relatório quando solicitado, armazenando dados de forma segura e permitindo acesso rápido e fácil.

1.1.1 Objetivo Geral

Desenvolver um sistema para web de gerenciamento dos encontros de orientação que permita a realização de um acesso que difere entre aluno e

professor, além do cadastro de novos encontros e a geração de um relatório dos encontros já cadastrados quando solicitado.

1.1.2 Objetivos específicos

- Estudar a linguagem de programação *Java* e suas tecnologias no desenvolvimento web.
- Analisar e desenvolver o sistema pretendido utilizando a linguagem de programação estudada.
- Utilizar as tecnologias *Java EE*, *Java Server Faces*, *Java Persistence API*, *EJB* e *Hibernate*.
- Estudar a utilização das bibliotecas de componentes *Primefaces* e *BootsFaces*.
- Utilizar o modelo *MVC* (*Model-View-Controller*) e o banco de dados *PostgreSQL*.

2. REFERENCIAL TEÓRICO

Este capítulo irá apresentar os trabalhos relacionados bem como conceitos necessários para o entendimento e desenvolvimento do sistema proposto.

2.1 Trabalhos relacionados

Nesta seção serão tratados alguns trabalhos relacionados com o presente trabalho.

2.1.1 FERRAMENTA MOODLE COMO RECURSO PARA GERENCIAMENTO E ORIENTAÇÃO DE TCC

O artigo produzido por Junior et. Al (2011) mostra uma análise do método tradicional de orientação e aponta fragilidades presentes nesse método, entre eles estão a falta de padrões no controle das atividades de orientação e a excessiva documentação produzida.

O sistema utilizado no artigo é o sistema MOODLE o qual já era utilizado em disciplinas a distância na Faculdade de Negócios e Tecnologias da Informação (FACNET). Após a análise da implementação do sistema para controle dessas orientações, foi constatado que não apenas o desempenho dos alunos melhorou, assim como o acompanhamento pelo professor orientador e coordenação do curso foi mais efetivo.

2.1.2 SISTEMA DE GERENCIAMENTO DE TCCS DO CURSO DE SISTEMAS DE INFORMAÇÃO DA UNIPLAC

O trabalho feito por Oliveira (2010) teve a finalidade de desenvolver um sistema *WEB* para o gerenciamento do processo de acompanhamento do TCC do Curso de Sistemas de Informação da Universidade do Planalto Catarinense (UNIPLAC), onde esse processo é feito manualmente através de formulários e relatórios, o objetivo final do trabalho foi, citando Oliveira (2010) “(...) tornando todo o processo de coordenação e acompanhamento dos projetos mais rápido e ágil, eliminando as redundâncias e o excesso de trabalhos manuais.”.

2.1.3 MODELAGEM DE UM SISTEMA DE INFORMAÇÃO APLICADO AO GERENCIAMENTO DO ACOMPANHAMENTO DE TRABALHOS DE CONCLUSÃO DE CURSO DO IFC-CAMPUS CAMBORIÚ

O trabalho feito por Maso (2016) baseado em Oliveira (2010) como referência para modelagem do sistema, analisando, alterando, incluindo ou removendo conceitos e funcionalidades quando necessário, e citando Maso (2016):

> Considerando que todo o processo de gestão do TC é realizado manualmente, propõe-se um sistema computacional que faça o gerenciamento do processo de gestão dos TCs, de forma a facilitar a comunicação, orientação e acompanhamento, além de organizar e facilitar o acesso aos documentos necessários durante o desenvolvimento dos trabalhos, suprindo a necessidade da constante interação entre os envolvidos.

Tal motivação resultou em um sistema mais enxuto que o apresentado por Oliveira (2010), esse sistema apresenta as funcionalidades: consulta de outros TCs já finalizados e aprovados, calendário de eventos da matéria.

O sistema proposto no presente trabalho possui de diferencial uma funcionalidade considerada crucial no contexto levantado pelo mesmo, à geração de relatório dos encontros registrados no sistema, bem como o controle do sistema diferenciando alunos e professores e limitando suas ações de acordo com esse controle.

2.2 TECNOLOGIAS

Nesta seção serão apresentadas as tecnologias utilizadas no desenvolvimento do sistema.

2.2.1 JAVA

A revolução dos microprocessadores tornou possível o desenvolvimento dos computadores pessoais, e vendo esse profundo impacto nos dispositivos de consumo popular a *Sun Microsystems*, em 1991, financiou um projeto de pesquisa que resultou em uma linguagem baseada em *C*++, nomeada de *Oak* por seu criador, James Gosling, em homenagem a uma árvore (carvalho) que era vista por sua janela na *Sun*. Foi descoberto mais tarde que já havia uma linguagem de computador com esse nome. O nome *Java* surgiu quando uma equipe da *Sun*

visitou uma cafeteria local, na qual o café era importado de uma cidade com esse nome (DEITEL E DEITEL, 2010).

A pesquisa passou por dificuldades, já que o mercado dos dispositivos eletrônicos destinados ao consumidor final não estava se desenvolvendo tão rapidamente quanto a *Sun* havia previsto. Em 1993 a *Web* explodiu em popularidade, então a *Sun* viu a oportunidade de utilizar o *Java* para adicionar conteúdo dinâmico, interatividade e animações, às páginas da *Web* (DEITEL E DEITEL, 2010).

O Java foi formalmente anunciado em 1995 em uma conferência do setor e chamou a atenção da comunidade por seu enorme interesse na Web. Atualmente o Java é utilizado para desenvolver aplicativos corporativos de grande porte, aprimorar a funcionalidade de servidores da Web, fornecer aplicativos voltados para o consumo popular e para muitos outros propósitos (DEITEL E DEITEL, 2010).

De acordo com *Oracle* (2016), a linguagem de programação *Java* é uma linguagem de alto nível que se destaca pelas seguintes características:

- Simples e Dinâmica;
- Orientada a objeto;
- Distribuída;
- Arquitetura Neutra;
- Portátil;
- Alto Desempenho;
- Robusta e segura.

O *Java* é bastante versátil, o que o torna multiplataforma. A *Sun* criou especificações para atender as distintas necessidades dos desenvolvedores, em conformidade com o tipo de plataforma: *desktop* (*Standard*), corporativa (*Enterprise*) e móvel (*Micro Edition*) (SAMPAIO, 2011).

A edição *Standard* (*Java SE- Standard Edition*) é usada na maioria das aplicações por possuir *Application Programming Interfaces* (*APIs* - Interface de Programação de Aplicativos) para os mais diversos problemas de programação, desde redes até chamadas remotas. A edição Enterprise (*Java EE - Enterprise Edition*) fornece especificações e *APIs* para *containers* (servidores) que hospedam aplicações corporativas. A edição Micro (*Java ME – Micro Edition*) fornece especificações, perfis de equipamentos e *APIs* para criação de programas para plataformas móveis ou embarcadas (SAMPAIO, 2011).

2.2.1.1 JAVA EE – Enterprise Edition

Segundo Sampaio (2011) o conceito do *Java EE* é que o desenvolvedor escreva menos código aproveitando as plataformas de serviços e *APIs* existentes. O modelo de programação do *Java EE* é baseado em containers, que fornecem todos os serviços necessários para a aplicação corporativa. Assim diminui o tempo de desenvolvimento, os riscos do projeto e os problemas de manutenção.

Para Sampaio (2011) pode-se resumir o *Java EE* em:

- *Framework*, serviços e *APIs* para desenvolvimento de aplicações corporativas;
- Implementado através de containers ou servidores de aplicação;
- *Web: JavaServerFaces, JavaServerPages* e *Servle*t;
- *EJB: Enterprise JavaBeans* e *Web Services*.

Ainda, para Sampaio (2011) os serviços e *APIs* do *Java EE* juntamente com o seu modelo de *containers* fornecem um ambiente completo para a execução de aplicações corporativas, assim o desenvolvedor pode se preocupar apenas com as funcionalidades desejadas e esse conjunto de recursos cuidará dos detalhes básicos.

2.2.1.2 JSF – JavaServer Faces

Segundo Gonçalves (2007) o *JSF* é uma tecnologia do *Java EE* e é projetado para tornar simples o desenvolvimento de aplicações *Web*. O *JSF* torna fácil o desenvolvimento através dos componentes de interface de usuário (*GUI - Graphical User Interface*).

O *JSF* é um framework baseado em componentes. Através do uso de componentes é possível desenvolver uma interface de usuário em um nível mais alto do que com o *HTML (HyperText Markup Language*) bruto. Para Geary e Horstmann (2012) o JSF é divido em partes:

- Um conjunto de componentes de *UI* (*User Interface*, ou interface do usuário) pré-fabricados;
- Um modelo de programação orientado a eventos;
- Um modelo de componentes que permite desenvolvedores independentes fornecerem componentes adicionais.

O *JSF* torna possível a escolha de fornecedores, pois é um padrão com múltiplas implementações. Outra vantagem se dá no fato que, além de estar continuamente em processo de aperfeiçoamento e atualização, o *framework* foi cuidadosamente analisado por um comitê de padronização (GEARY e HORSTMANN, 2012).

2.2.2 Java Persistence API - JPA

A *JPA* é baseada no conceito de objeto simples, que incorpora ideias de frameworks de persistência para padronizar o mapeamento objeto relacional em *Java*. Na *JPA* os objetos persistentes são denominados entidades (*Entities*). Uma entidade é um objeto simples, que representa um conjunto de dados persistido no banco de dados. Uma vez que entidades são definidas por classes *Java* comuns, sem relações com *frameworks* e bibliotecas, elas podem ser abstratas ou herdar de outras classes. Para que uma entidade se torne persistente, é necessário associa-la a um contexto de persistência, esse que fornece a conexão entre as instâncias e o banco de dados (GONÇALVES, 2007).

As classes e *interfaces* da *JPA* estão localizadas no pacote *javax.persistence*. Com isso, é possível fazer o mapeamento da aplicação utilizando anotações, assim pode-se dispensar os descritores *XML* para cada entidade da aplicação. Assim, uma entidade é rotulada com a anotação *@Entity*, sendo ela uma classe *Java* comum. Uma tabela é representada pela anotação *@Table* e a chave primária pela anotação *@Id* e cada coluna é especificada pela anotação *@Column* (GONÇALVES, 2007).

2.2.2.1 Anotações Java – *Java Annotation*

Java Annotation são tipos de dados especialmente definidos com o intuito de simplificar tarefas em Java com uma simples anotação, colocada em frente à (ou acima de) elementos como classes, métodos, campos e variáveis. Quando um elemento do programa é anotado, o compilador lê a informação contida nessa anotação e pode reter essa informação nos arquivos de classe ou dispor disso de acordo com o que foi especificado na definição do tipo da anotação. Assim, citando GOLÇALVES (2007):

> Quando retiver nos arquivos de classe os elementos contidos na anotação podem ser examinados em runtime por uma API baseada em reflexão. Dessa forma, o JVM (Java Virtual Machine) ou outros programas podem olhar esse metadata

> para determinar como interagir com os elementos do programa ou alterar seus comportamentos.

Anotações é uma forma de se opor a rigidez da herança, enriquecendo uma classe com comportamentos de forma mais independente. Uma anotação é precedida, sendo ela personalizada ou não, por um símbolo de @ seguida de uma meta-anotação (GONÇALVEZ, 2007).

2.2.3 Hibernate

O *Hibernate* é um *framework* com o objetivo de dar uma completa solução para o complicado gerenciamento de dados persistentes em *Java*. O *framework* se relaciona com o banco de dados, relacionamento esse conhecido como mapeamento objeto/relacional (*ORM - Object-relational Mapping*), assim o desenvolver pode se concentrar em problemas da lógica do negócio. A escrita da lógica de negócios e classes persistentes devem seguir algumas regras como padrão de desenvolvimento. O *Hibernate* se integra suavemente ao sistema, e se comunica com o banco de dados como se fosse diretamente da aplicação, assim mesmo uma mudança de banco de dados pode ser feito através de alguns detalhes na configuração do *Hibernate* (GONÇALVES, 2007).

2.2.4 Enterprise JavaBean - EJB

Para Roman et. Al (2008) o padrão *EJB* é uma arquitetura de componentes para componentes instaláveis do lado do servidor em *Java*. É um acordo entre componentes e servidores de aplicação que permite a qualquer componente executar em qualquer um desses servidores. Os *enterprise Beans* são instaláveis então podem ser importados e carregados em um servidor de aplicação, que os hospeda.

Embora os componentes *EJB* só devam ser escritos em linguagem *Java*, a linguagem é ideal para construí-los por muitas razoes:

- **Separação entre interface e implementação:** precisa-se de uma separação clara entre interface e implementação para comercializar componentes. O *Java* consegue fazer o suporte disso através das palavras-chave *interface* e *class.*
- **Segurança e proteção:** a arquitetura *Java* pode ser considerada mais segura do que muitas linguagens de programação. Em *Java*, se uma *thread* “morrer”, a aplicação permanecerá ativa. Ponteiros não são mais um problema. Vazamentos de

memória ocorrem com menor frequência. E também o *Java* possui um rico conjunto de bibliotecas, de modo que não é só a sintaxe de uma linguagem, mas um conjunto inteiro de bibliotecas depuradas pré-escritas que permite aos desenvolvedores evitar a reinvenção da roda de maneira falha.

- **Multiplataforma:** o *Java* executa em qualquer plataforma. Como o *EJB* é uma aplicação de *Java*, isso significa também que deve executar facilmente em qualquer plataforma. (ROMAN et. Al, 2008).

Segundo Roman et. AL (2008) especificamente o *EJB* é utilizado para ajudar a resolver problemas de negócio, uma vez que seus componentes podem realizar várias tarefas:

- **Realização da lógica do negócio:** por exemplo, enviar um correio eletrônico de confirmação usando *JavaMail API*.
- **Acesso a um banco de dados:** os *enterprise Beans* conseguem acesso ao banco de dados utilizando a *API Java Database Connectivity (JDBC)*.
- **Acesso a outro sistema:** os enterprise Beans conseguem integrar uma aplicação por meio do Java Connector Architecture (JCA).

Segundo Roman et. AL (2008) os componentes *EJB* não são *GUIs*, os *enterprise beans* permanecem atrás das *GUIs* e fazem todo o trabalho duro. Exemplos de GUIs que podem se conectar aos *enterprise beans:*

- **Clientes pesados:** os clientes pesados executam em um computador do usuário. Eles podem se conectar por meio da rede com componentes *EJB* que residem em um servidor. Esses componentes podem realizar qualquer uma das tarefas (lógica de negócio, lógica de banco de dados ou acesso a outros sistemas). Os clientes pesados em *Java* incluem *applets* (mini aplicações) e aplicações.
- **Páginas da Web dinamicamente geradas:** tratam-se de sites da *Web* complexos que precisam que suas páginas sejam geradas especificamente para cada solicitação. Por exemplo, a *home page* da *Amazon.com* é completamente diferente para cada usuário, dependendo do perfil dos mesmos. Os *servlets* de *Jav*a e as *JSPs* são utilizadas para gerar essas páginas especificas, esses que por sua vez vivem dentro de um servidor *Web* e podem se conectar a componentes de *EJB*, gerando páginas diferentes baseadas nos valores retornados da camada de *EJB*.

O modelo de programação dos *EJB's* é eficiente harmonizando simplicidade no seu uso com robustez, esse modelo de desenvolvimento permite a reutilização e

escalabilidade do código, através de anotações utilizadas num *container*. No *EJB* um *container* é um ambiente de tempo de execução que disponibiliza serviços, sendo eles, gerenciamento de transações, controle de concorrência, agregação e autorização de segurança. Com isso os desenvolvedores conseguem dedicar-se somente na implementação da lógica, durante o tempo que o *container* cuida das conexões técnicas (GONÇALVES, 2013).

Segundo Gonçalves (2013) existem os seguintes *beans* de sessão:

- **Sem estado:** Não possui nenhum estado de conversação para um cliente específico, podendo ser utilizada por qualquer cliente, podendo ser definida pela anotação @*Stateless*;
- **Com estado:** Possui o estado de conversação para um cliente específico, podendo ser definido usando a anotação @*Stateful*;
- **Singular:** Contém somente um *bean* de sessão, fornece facilidade ao acesso concorrente, podendo ser definido através da anotação @*Singleton*.

2.2.5 Biblioteca de Componentes

Nesta seção serão apresentadas algumas bibliotecas de componentes que auxiliarão o *JSF* na criação dos elementos visuais.

2.2.5.1 BootsFaces

O BootsFaces é uma biblioteca que agrega elementos visuais ao *JavaServer Faces*, baseia-se no estilo visual do *framework Bootstrap*. O Bootsfaces é disponibilizado em um arquivo *.jar*, possui suporte à aplicação de *AJAX* e permite integração com outros *frameworks* como por exemplo o *PrimeFaces*, o *OmniFaces* e o *ButterFaces* (BOOTSFACES, 2017).

Com a biblioteca *BootsFaces* torna-se possível utilizar componentes previamente preparados para que a interface com o usuário se adapte automaticamente de acordo com o dispositivo utilizado, por exemplo: *desktop* ou *mobile*, pois, essa biblioteca permite o desenvolvimento de *layout* responsivo para as aplicações (BOOTSFACES, 2017).

2.2.5.2 Primefaces

Os componentes do *PrimeFaces* são desenvolvidos com um princípio de criação que afirma que “Um bom componente de *interface* deve ocultar a complexidade, mas manter a flexibilidade” ao fazê-lo. (PRIMEFACES, 2018).

O *PrimeFaces* a exemplo do *BootsFaces* é disponibilizado por meio de um arquivo com a extensão *.jar*, dessa forma não necessita de dependências ou configurações adicionais para trabalhar. Os elementos dessa biblioteca possuem funcionalidades de *Ajax* integrado por padrão, fundamentado na *API* de *Ajax* do *JavaServer Faces*.

Com a biblioteca *PrimeFaces* torna-se menos complexa a tarefa de implementar uma interface gráfica para sistemas web baseados em *JavaServer Faces*, pois com a utilização de seus componentes abstrai-se a complexidade envolvida na criação e organização do design de uma *interface* com o usuário. Conforme a *DevRates* (Figura 1) – de acordo com divulgação no *site* do *PrimeFaces*–, o *PrimeFaces* assumiu a liderança como o *framework* favorito para criação de *interfaces* pelos desenvolvedores pesquisados. (PRIMEFACES, 2018).

Figura 1 – *Frameworks* favoritos dos desenvolvedores

Fonte: PRIMEFACES, 2017

2.2.6 Model-View-Controller - MVC

Conforme Gonçalves (2013), o padrão de projeto modelo-visão-controlador (*MVC*) é um padrão utilizado para separar a lógica funcional da interface de usuário. Com a utilização do *MVC* pode-se modificar tanto a interface quanto as regras de negócio sem que uma afete a outra, isso se torna possível devido à separação de conceitos. No *MVC*, o modelo representa os dados da aplicação, a visualização

refere-se à *interface* com o usuário, e o controlador gerencia a comunicação entre ambos.

2.2.7 PostgreSQL

Para Milani (2008) o *PostgreSQL* é um Sistema Gerenciador de Banco de Dados (*SGBD*) Relacional, e é utilizado para armazenar e administrar o acesso a informações de soluções de informática em todas as áreas de negócios existentes.

As soluções de informática estão presentes em diversas áreas de negócios, desde a própria tecnologia da informação, bem como em áreas como o setor aéreo, bancário, governamental, automobilístico, etc. Uma das propostas do *PostgreSQL* é que, toda e qualquer aplicação que armazene dados para uso ou acesso posterior está, de alguma forma, integrada a um banco de dados, seja armazenamento em memória, arquivos de texto, ou em tabelas. (MILANI, 2008).

Apenas o armazenamento de informações não torna o mecanismo utilizado em um *SGBD*. Se todas as informações de uma solução forem armazenadas em um único arquivo de texto, qualquer pessoa que tenha acesso de leitura a este arquivo poderá consulta-lo por inteiro, e não apenas a informação que desejar, violando, assim as leis de sigilo envolvidas nas aplicações. O mesmo ocorre se a pessoa tiver acesso de escrita no arquivo de texto em questão, pois poderá alterar qualquer informação, mesmo a que não lhe pertence. Um *SGBD* deve controlar, além do armazenamento de dados, o seu acesso – quem pode ler cada dado e quem pode alterar cada informação. (MILANI, 2008).

Em termos básicos, o *PostgreSQL* é uma ferramenta encarregada de armazenar dados e gerenciar o acesso a cada informação de acordo com regras previamente definidas.

2.2.8 JasperReports

JasperReports é uma biblioteca *Java open-source* para o desenvolvimento de relatórios, ela não é uma aplicação autônoma, deve-se incluir sua biblioteca no *CLASSPATH* da aplicação, pode ser utilizada em qualquer aplicação *Java*, incluindo *desktops*, *Web* e distribuídas. (HEFFELFINGER, 2009).

Apresenta grande habilidade na organização e apresentação de conteúdo, permitindo a geração dinâmica de relatórios em diversos formatos, tais como, *PDF (Portable Document Format), HTML, XLS, CSV e XML*.

3. METODOLOGIA

Este capítulo irá tratar do estudo de caso feito, o contexto atual, a nova solução proposta, bem como a modelagem do sistema de acordo com os diagramas da *UML* (*Unified Modeling Language* – Linguagem de Modelagem Unificada).

Inicialmente foi realizado um estudo para se obter conhecimento sobre as tecnologias para o desenvolvimento do trabalho. Os dados necessários ao desenvolvimento do sistema foram retirados da ficha de encontro (documento disponibilizado pelo IFSUL campus Passo Fundo), conforme Anexo A.

3.1 Estudo de Caso

O objetivo do trabalho proposto é o desenvolvimento de uma solução para o gerenciamento dos encontros de orientação. Nesta seção será apresentado o contexto atual de como é feito a documentação desses encontros, bem como a proposta da nova solução.

3.2 Contexto Atual

O gerenciamento dos encontros de orientação é feito através de um documento disponibilizado pela instituição, neste contém alguns dados sobre o aluno, o professor orientador e co-orientador (se houver), o titulo do TC, e o registro dos encontros propriamente ditos, que contêm data, hora, os assuntos tratados e os vistos do orientador e orientando.

Atualmente tal documentação é feita de maneira manual, geralmente, em documentos eletrônicos ou em papéis impressos, e fica a cargo do orientador. Analisando o contexto atual, o mesmo apresenta algumas falhas nesse gerenciamento como, por exemplo, a falha de comunicação entre orientando e orientador no que diz respeito aos assuntos tratados na atual orientação e os combinados para a próxima, ou então, no caso de papel impresso, a perda de tal papel resultará em uma dificuldade de recuperar as informações contidas nele.

3.3 Proposta de nova solução

A solução proposta será desenvolvida em formato de uma aplicação *WEB* com *design* responsivo, isto é, adaptar o *layout* da página de acordo com a

resolução de visualização, e por se tratar de uma aplicação *WEB* qualquer usuário autorizado e conectado a internet poderá acessar o sistema.

Na solução proposta, os dados relativos às orientações serão salvos no *SGBD PostgreSQL*. O sistema terá controle de acesso diferenciando entre orientador e orientando, assim cada tipo de usuário terá acesso ao sistema de acordo com suas permissões. Pela parte do aluno, esse poderá enviar o documento da atual versão do TC com uma mensagem informando as alterações feitas, consultar as informações das orientações e gerar o relatório das informações atuais no modelo do documento disponibilizado. O professor poderá criar alterar ou excluir um novo documento de orientação, inserir e alterar informações pertinentes às orientações, bem como data, hora, e assuntos, além de poder enviar um documento com a versão revisada do TC e gerar um relatório.

3.4 Requisitos Funcionais e Não Funcionais

Os requisitos funcionais definem as funções e serviços do sistema. Os requisitos não funcionais definem as propriedades e restrições do sistema, tendo objetivo tornar o sistema mais seguro.

Estes requisitos foram definidos através do estudo de caso.

3.4.1 Requisitos Funcionais

- Usar um sistema de *login* e senha para identificar o usuário;
- Manter Usuários;
- Manter Dados;
- Manter Curso;
- Manter Tipo;
- Manter Ficha;
- Manter Encontros;
- Enviar Versão;
- Enviar Revisão;
- Gerar Relatório.

3.4.2 Requisitos Não Funcionais

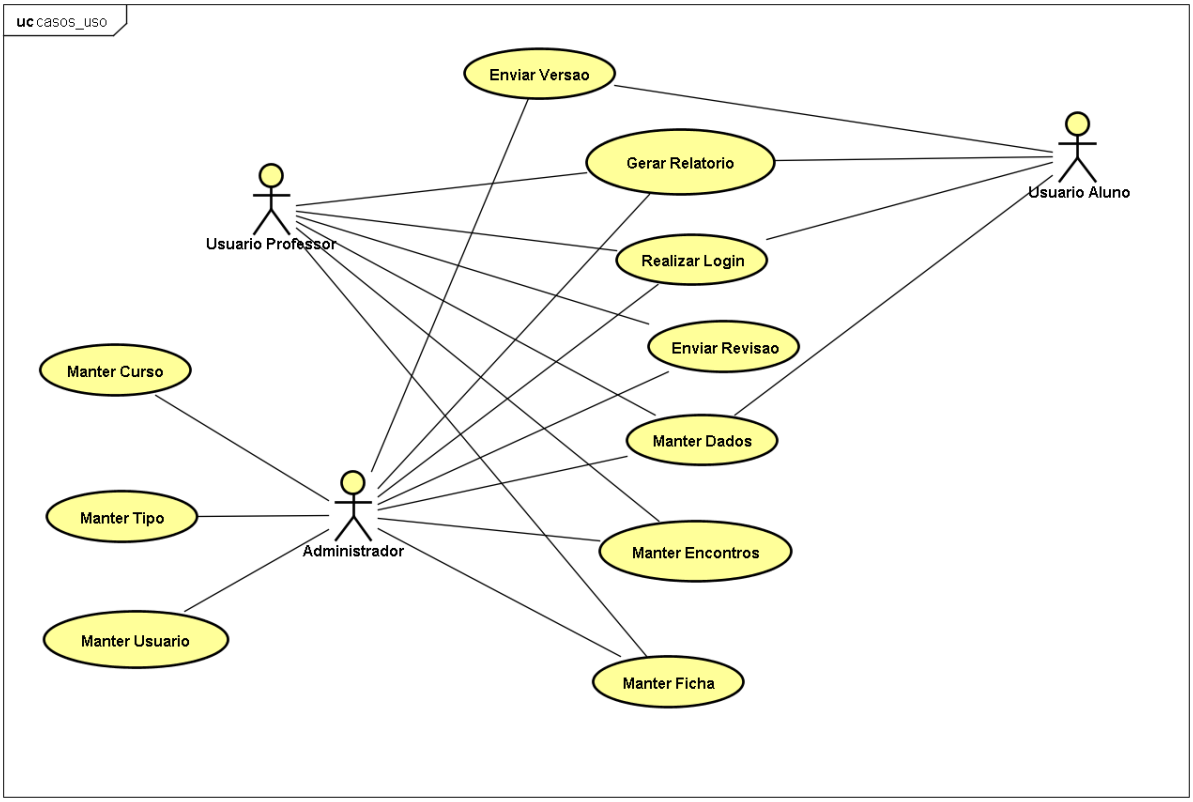
- A *interface* deve ser responsiva e intuitiva;

- O sistema deve tratar acessos não autorizados;
- A persistência dos dados deve ser feita em um *SGBD*.

3.5 Diagrama de Casos de Uso

O diagrama de casos de uso (Figura 2) foi elaborado com base no estudo de caso para apresentar as ações dos usuários professor (sendo um administrador do sistema) e aluno, assim como as funcionalidades do sistema. Ambos os usuários deverão realizar o *login*, permitindo assim fazer suas respectivas ações no sistema.

Figura 2 – Diagrama de casos de uso



Fonte: do Autor.

3.6 Descrição dos Casos de Uso

Esta seção descreverá os passos de alguns casos de uso de forma descritiva, levando em consideração as pré-condições, o fluxo de ações, entre outros.

3.6.1 Caso de Uso Manter Usuário

O caso de uso Manter Usuário (Tabela 1) é bem semelhante ao Manter Tipo e Manter Curso, todos são realizados pelo professor que será responsável pela

administração do sistema. A visualização dos dados já cadastrados se dá em forma de lista no acesso a página de administração de cada conjunto de dados.

Tabela 1 – Descrição Caso de Uso Manter Usuário

Nome do Caso de Uso	**Manter Usuário**
Caso de Uso Geral	
Ator Principal	Professor
Atores Secundários	
Resumo	Este caso de uso descreve as ações para criar, alterar e excluir novos usuários do sistema.
Pré-condições	Exigência de novo usuário do sistema.
Pós-condições	Usuário persistido.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Acessar a área de administração de Usuário e solicitar a criação de novo usuário.	
	2. Mostrar o formulário para preenchimento dos dados.
3. Informar os dados obrigatórios e clicar no botão Salvar Dados.	
	4. Salvar os dados no SGBD
Restrições/Validações/Regras de Negócio	1. O campo *user* deve ser único no sistema. 2. O Professor deve ter permissão de administrador.
Fluxo Alternativo I – Edição de Usuário	
Ações do Ator	**Ações do Sistema**
1. Acessar a área de administração de Usuário e solicitar a edição do usuário na lista de visualização.	
	2. Mostrar o formulário com os dados previamente cadastrados para alteração dos dados.
3. Fazer as alterações e clicar no botão	

Salvar Dados.	
	4. Salvar os dados no SGBD.
Restrições/Validações/Regras de Negócio	1. O Professor deve ter permissão de administrador.
Fluxo Alternativo II – Remover Usuário	
Ações do Ator	**Ações do Sistema**
1. Na lista de visualização, clicar no ícone de lixeira na coluna de Ações.	
	2. Remover usuário do SGBD e atualizar a lista de visualização.
Restrições/Validações/Regras de Negócio	1. O Professor deve ter permissão de administrador.

3.6.2 Caso de Uso Realizar Login

O caso de uso Realizar Login (Tabela 2) é um dos mais importantes no sistema, nele é feito toda a parte da segurança do acesso aos dados de acordo com as permissões vinculadas ao usuário autenticado.

Tabela 2 – Descrição Caso de Uso Realizar Login

Nome do Caso de Uso	**Realizar Login**
Caso de Uso Geral	
Ator Principal	Professor, Aluno
Atores Secundários	
Resumo	Este caso de uso descreve as ações para realizar o *login* no sistema.
Pré-condições	Usuário já estar cadastrado no sistema.
Pós-condições	Acesso ao sistema com as permissões vinculadas ao usuário autenticado.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Usuário informa o *user* e a senha, e clica no botão de *Login.*	
	2. Sistema autentica os dados informados e redireciona o usuário para a página inicial.
Restrições/Validações/Regras de	1. O sistema deve dar acesso apenas

Negócio	às permissões vinculadas ao usuário autenticado.
Fluxo Alternativo – Usuário Não Cadastrado	
Ações do Ator	**Ações do Sistema**
1. Usuário informa dados não cadastrados ou incorretos.	
	2. Sistema informa erro no *user* ou senha, e/ou para solicitar o cadastro ao administrador.
Restrições/Validações/Regras de Negócio	1. *User* deve ser único no sistema.

3.6.3 Caso de Uso Gerar Relatório

O caso de uso Gerar Relatório (Tabela 3) permite ao professor e aluno gerar um *PDF* no formato disponibilizado, pelo IFSUL Campus Passo Fundo, dos encontros de orientação, contendo os dados pertinentes já cadastrados.

Tabela 3 – Descrição Caso de Uso Gerar Relatório

Nome do Caso de Uso	**Gerar Relatório**
Caso de Uso Geral	
Ator Principal	Professor, Aluno
Atores Secundários	
Resumo	Este caso de uso descreve as ações para gerar um relatório no formato PDF.
Pré-condições	Possuir permissão para gerar o relatório.
Pós-condições	Visualização do relatório pronto no formato PDF.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Usuário acessa a ficha de encontro e clica botão Gerar Relatório.	
	2. Sistema abrirá em uma nova aba do navegador o relatório, dos dados cadastrados na Ficha, no formato PDF.
Restrições/Validações/Regras de	2. O relatório deve ser no formato

Negócio	disponibilizado pelo Instituto.

3.6.4 Caso de Uso Manter Ficha

O caso de uso Manter Ficha (Tabela 4) descreve as possíveis ações do Professor para a manutenção da Ficha, que mostrará todos os dados pertinentes aos encontros de orientação.

Tabela 4 – Descrição Caso de Uso Manter Ficha

Nome do Caso de Uso	**Manter Ficha**
Caso de Uso Geral	
Ator Principal	Professor
Atores Secundários	
Resumo	Este caso de uso descreve as ações para a manutenção da classe Ficha.
Pré-condições	Ter um novo aluno sob orientação para criação da nova ficha.
Pós-condições	Persistir nova Ficha no SGBD.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Professor entra na página de Ficha, e clica no botão para a criação de nova Ficha.	
	2. Sistema abre formulário para preenchimento dos dados.
3. Professor informa os dados necessários e clica no botão salvar.	
	4. Sistema salva nova ficha no SGBD e atualiza a lista de visualização.
Restrições/Validações/Regras de Negócio	1. Ser um Usuário Professor
Fluxo Alternativo I – Edição de Ficha	
Ações do Ator	**Ações do Sistema**
1. Acessar página de Ficha e solicitar a edição da Ficha na lista de visualização.	
	2. Mostrar o formulário com os dados

	previamente cadastrados para alteração dos dados.
3. Fazer as alterações e clicar no botão Salvar Dados.	
	4. Salvar os dados no SGBD e atualizar a lista de visualização.
Restrições/Validações/Regras de Negócio	1. Ser um Usuário Professor.
Fluxo Alternativo II – Remover Ficha	
Ações do Ator	**Ações do Sistema**
1. Na lista de visualização, clicar no ícone de lixeira na coluna de Ações.	
	2. Remover a ficha do SGBD e atualizar a lista de visualização.
Restrições/Validações/Regras de Negócio	1. Ser um Usuário Professor.

3.6.5 Caso de Uso Enviar Versão

O caso de uso Enviar Versão (Tabela 5) é igual ao caso de uso Enviar Revisão, a diferença é apenas que um é de Aluno e outro do Professor, respectivamente. Os documentos serão salvos em uma lista anexada à ficha, conseguindo assim fazer o acompanhamento das versões ao longo do desenvolvimento do TC.

Tabela 5 – Descrição Caso de Uso Enviar Versão

Nome do Caso de Uso	Enviar Versão
Caso de Uso Geral	
Ator Principal	Aluno
Atores Secundários	
Resumo	Este caso de uso descreve as ações para enviar a versão atual do documento do PC.
Pré-condições	Possuir permissão para envio do documento.
Pós-condições	Persistência dos dados no SGBD.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**

1. Aluno acessa a página de Ficha, e clica no botão para adicionar novo documento.	
	2. Sistema abrirá o formulário para a inclusão do documento.
3. Aluno seleciona o documento e clica em salvar.	
	4. Sistema salva o documento, em conjunto com a data de envio no SGBD.
Restrições/Validações/Regras de Negócio	1. A data deve ser pega automaticamente do sistema operacional.

3.6.6 Caso de Uso Manter Encontros

O caso de uso Manter Encontro (Tabela 6) descreve as possíveis ações do Professor para a manutenção dos Encontros, que mostrará todos os dados pertinentes dos encontros de orientação.

Tabela 6 – Descrição Caso de Uso Manter Encontros

Nome do Caso de Uso	**Manter Encontros**
Caso de Uso Geral	
Ator Principal	Professor
Atores Secundários	
Resumo	Este caso de uso descreve as ações para a manutenção da classe Encontros.
Pré-condições	Possuir permissão de professor para manutenção, possuir uma Ficha previamente criada.
Pós-condições	Persistir novo encontro no SGBD e atualizar a lista dos encontros na Ficha.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
5. Professor entra na página de Ficha, seleciona a Ficha pretendida e clica no botão para adicionar um novo	

Encontro.	
	6. Sistema abre formulário para preenchimento dos dados.
7. Professor informa os dados necessários e clica no botão salvar.	
	8. Sistema salva o novo Encontro no SGBD e atualiza a lista de visualização.
Restrições/Validações/Regras de Negócio	
Fluxo Alternativo I – Edição de Encontros	
Ações do Ator	**Ações do Sistema**
5. Acessar página da Ficha pretendida e solicitar a edição do Encontro na lista de visualização dos Encontros.	
	6. Mostrar o formulário com os dados previamente cadastrados para alteração dos dados.
7. Fazer as alterações e clicar no botão Salvar Dados.	
	8. Salvar os dados no SGBD e atualizar a lista de visualização.
Fluxo Alternativo II – Remover Encontro	
Ações do Ator	**Ações do Sistema**
3. Na lista de visualização, clicar no ícone de lixeira na coluna de Ações.	
	4. Remover o Encontro do SGBD e atualizar a lista de visualização.

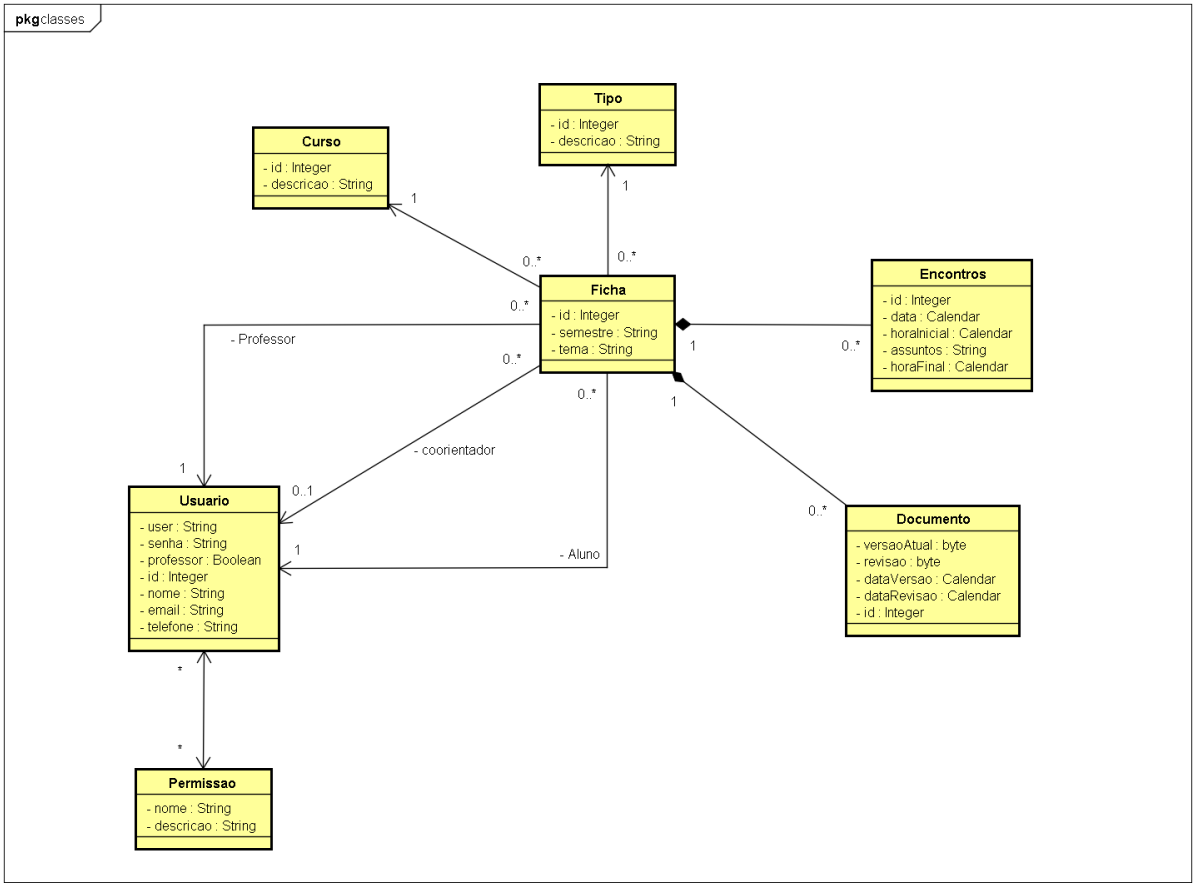
3.7 Diagrama de Classes

O diagrama de classes apresentado na Figura 3 foi desenvolvido para demonstrar as classes, atributos e relacionamentos do sistema.

A classe Usuário possui os atributos de *user* e *senha* não necessitando do atributo *id, pois* o atributo *user* será único no sistema, o atributo *professor* serve para distinguir os usuários entre professor e aluno.

A classe Ficha receberá por seus relacionamentos um aluno e um professor, esse o qual terá acesso para adicionar novas instâncias da classe Encontros - que constará com os atributos de *data, hora* e *assuntos* -, na classe Ficha também pode ser adicionado um co-orientador.

Figura 3 – Diagrama de classes



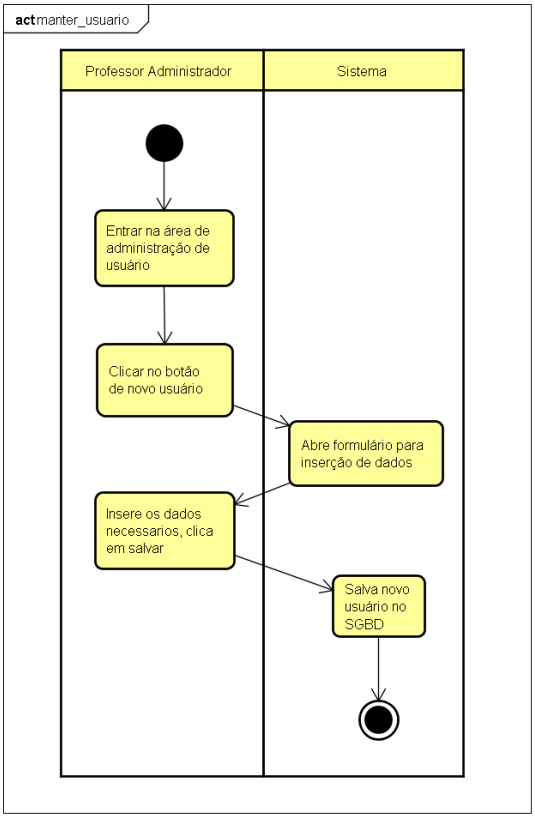
Fonte: do Autor.

3.8 Diagrama de Atividade

Os diagramas de atividade são essenciais para demonstrar as ações envolvidas por cada parte em um determinado processo do sistema.

O diagrama ilustrado na Figura 4 demonstra o processo para cadastro de um novo usuário.

Figura 4 – Diagrama de atividade cadastro de usuário



Fonte: do Autor.

O sistema se baseia em ações simples como na ilustração do diagrama apresentado, algumas ações, tal como excluir dados, possuem alguns passos a menos, assim o sistema se torna menos complexo.

4. DESENVOLVIMENTO

Este capítulo descreve o processo utilizado para o desenvolvimento da solução proposta, a estrutura das camadas da aplicação e os recursos utilizados para o seu funcionamento.

4.1 FERRAMENTAS DE DESENVOLVIMENTO E RECURSOS

Esta seção apresenta as ferramentas e recursos utilizados no desenvolvimento da aplicação.

4.1.1 *IDE (Integrated Development Environment)* e Servidores

Para o desenvolvimento do sistema foi utilizado uma aplicação *IDE* (Ambiente de Desenvolvimento Integrado), a qual foi necessário para a escrita do código. Como servidor da aplicação utilizou-se o *GlassFish* versão 4.1, esse mantido pela empresa *Oracle* possui seu código fonte aberto e suporta todas as especificações da *API Java Enterprise Edition*. A *IDE* escolhida foi *o NetBeans IDE 8.2*, por sua compatibilidade com servidor *GlassFish*.

O gerenciamento do banco de dados foi feito através do SGBD PostgreSQL. Por meio do servidor desse sistema foi possível guardar os dados e comunicar-se com a aplicação web desenvolvida.

4.1.2 Bibliotecas e *Frameworks*

Para a construção do sistema foram utilizados recursos de diferentes bibliotecas e *frameworks*, que trabalham em conjunto com o *JSF*. Os recursos foram aplicados para auxiliar na criação da interface de usuário, para validar e persistir os dados e tratar da comunicação com a aplicação, entre o banco de dados e o servidor. As bibliotecas empregadas na aplicação são:

- **Biblioteca *Hibernate Validator 5.2.4***: foi utilizada para validar os dados vinculados ao objeto passado para persistência no banco, através de anotações foi possível definir formatos e restrições para cada atributo vinculado ao objeto passado, assim prevenindo que erros nos dados sejam persistidos no banco de dados.

- **Biblioteca *Hibernate JPA 4.3.11 JPA 2.1***: A biblioteca *Hibernate JPA* é uma biblioteca de persistência de dados que implementa a especificação *Java Persistence API* e foi utilizada para realizar o mapeamento objeto-relacional de objetos.

- **Biblioteca *JasperReports***: A biblioteca *JasperReports* foi utilizada como solução para a geração de relatórios em *Java*, com esse *framework* tornou-se possível criar um modelo de acordo com o especificado pela Instituição, com o objetivo de gerar relatórios dos encontros em formato *PDF*.

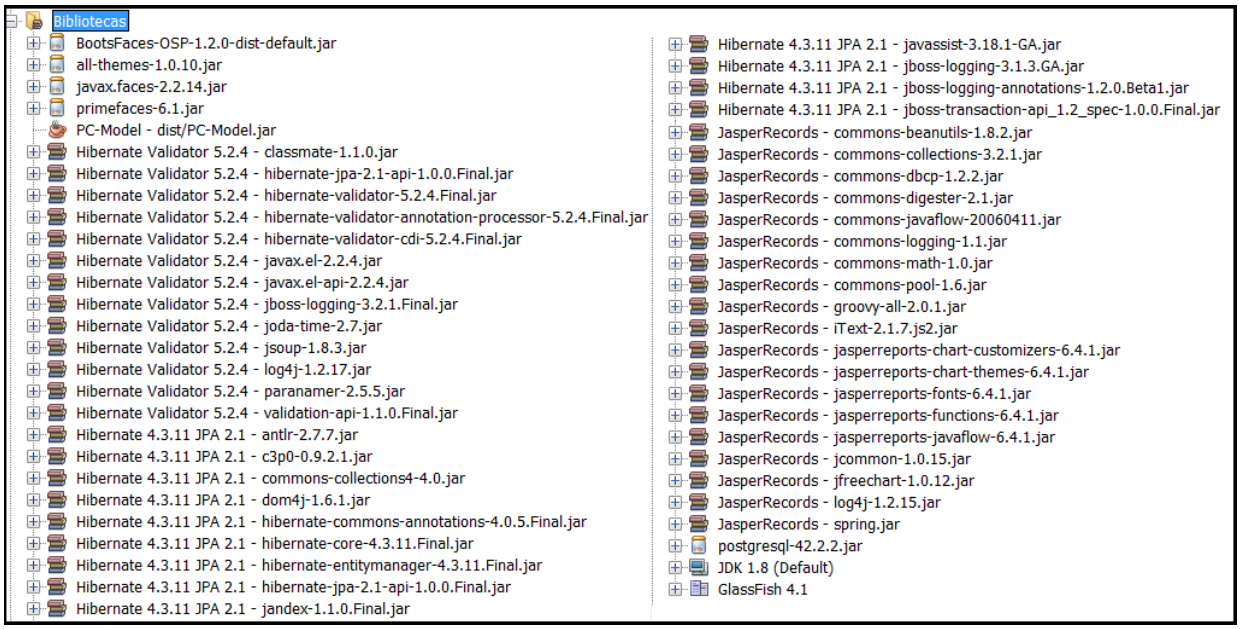
- **Biblioteca *BootsFaces 1.2.0***: A biblioteca *BootsFaces* é uma biblioteca de recursos gráficos para o *JSF* fundamentada no estilo visual da biblioteca *Bootstrap*. A finalidade de empregar essa biblioteca ao sistema consiste em que seus recursos são elaborados para a utilização em *layout* responsivo, sendo este um dos objetivos da aplicação *web*.

- **Biblioteca *PrimeFaces 6.1***: A biblioteca *PrimeFaces* possui como sua principal funcionalidade, auxiliar na implementação de recursos gráficos. Essa biblioteca foi utilizada no projeto em conjunto com a biblioteca *BootsFaces*.

- ***Driver JDBC PostgreSQL* versão 42.2.2**: O *driver JDBC* do *PostgreSQL* dispõe de elementos para a conexão do sistema *web* com o banco de dados, possibilitando gerenciar e manipular o banco de dados da aplicação.

Na Figura 5 podem ser visualizadas as bibliotecas utilizadas no sistema.

Figura 5 – Bibliotecas Utilizadas na Aplicação



Fonte: do Autor.

4.2 Estrutura da aplicação

Esta seção apresenta a arquitetura detalhada das camadas da aplicação, e os componentes que foram utilizados na criação do sistema.

4.2.1 Camada de Modelo

Essa camada é responsável pela lógica, responsável pela regra do negócio, essa camada representa uma das partes mais importantes do *software*. O modelo encapsula o estado e o comportamento da aplicação além de ser o único componente do *MVC* que faz interface da aplicação frente à fonte de dados, que é normalmente representada pelo banco de dados da aplicação.

O mapeamento objeto-relacional foi feito pela *JPA*, para realizar a persistência das classes no banco de dados, essas classes além de seguir o modelo *JavaBeans* - isto é, implementar a *interface Serializable* e possui atributos encapsulados, os quais são modificados através dos métodos *getter* e *setter* -, também são validadas com a ajuda do *Hibernate*, usando o conjunto de anotações tanto da *JPA* quanto do *Hibernate* para evitar que dados inválidos sejam persistidos no *SGBD*. A Figura 6 mostra um fragmento de código da classe *Ficha.*

Figura 6 – Fragmento de código da classe *Ficha*

```
@Entity
@Table(name = "ficha")
public class Ficha implements Serializable {

    @Id
    @SequenceGenerator(name = "seq_projeto", sequenceName = "seq_projeto_id", allocationSize = 1)
    @GeneratedValue(generator = "seq_projeto", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @NotNull(message = "O semestre não pode ser nulo")
    @NotBlank(message = "O semestre não pode ser em branco")
    @Length(max = 15, message = "O semestre não pode ter mais que {max} caracteres")
    @Column(name = "semestre", length = 15, nullable = false)
    private String semestre;
```

Fonte: do Autor.

São as anotações que definem as propriedades do objeto, bem como suas validações. A primeira anotação que podemos observar é a *@Entity*, ela é responsável por definir essa classe como uma tabela no banco de dados, onde o nome dela será definido pela anotação *@Table* seguido do atributo *name*= “nome_tabela”. O atributo *@Id* define a chave primária da tabela, podendo esse ser gerado de maneira automática com as anotações *@SequenceGenerator*, que gera esse campo automaticamente e define o tamanho do incremento, e *@GeneratedValue* que define a estratégia de geração como sequencial para o gerador criado posteriormente através do atributo *strategy*= *GenerationType.SEQUENCE,* esses são ligados pelo nome dado no gerador através do atributo *name*. A anotação *@Column* é usada para definir as colunas de uma tabela do banco de dados, alguns atributos podem ser usados nela para fazer validações de inserção de dados direto no *SGBD*, ao invés da interface do sistema, alguns deles são *name* que define o nome da coluna, *length* para o tamanho máximo dos dados aceitos e *nullable* para não permitir dados nulos.

Algumas outras anotações são usadas para validação dos dados e para fazer o relacionamento entre as tabelas, podemos observar algumas no fragmento do código da classe *Encontro* na Figura 7.

Figura 7 – Fragmento de código da classe *Encontro*

```
@NotNull(message = "Data não pode ser nula")
@Temporal(TemporalType.DATE)
@Column(name = "dia", nullable = false)
private Calendar dia;
@NotNull(message = "Data não pode ser nula")
@Column(name = "inicial", nullable = false, columnDefinition = "time")
private Calendar horaInicial;
@NotNull(message = "O campo assuntos não pode ser nulo")
@NotBlank(message = "O campo assuntos não pode ser em branco")
@Column(name = "assuntos", nullable = false, columnDefinition = "text")
private String assuntos;
@NotNull(message = "Data não pode ser nula")
@Column(name = "final", nullable = false, columnDefinition = "time")
private Calendar horaFinal;
@NotNull(message = "A ficha não pode ser nula")
@ManyToOne
@JoinColumn(name = "ficha", referencedColumnName = "id", nullable = false,
        foreignKey = @ForeignKey(name = "fk_encontro_ficha"))
private Ficha ficha;
```

Fonte: do Autor.

A anotação *@NotNull* define que não poderão ser aceitos valores nulos neste campo, *@Temporal* seguida de *TemporalType.DATE* define que apenas datas são aceitas nesse campo, *@NotBlank* não permite que dados em brancos sejam salvos e *@Length* mostrado na Figura 7 define um número máximo de caracteres aceitos naquele campo. Destas anotações citadas com exceção da anotação *@Temporal* todas permitem uma mensagem a ser mostrada no caso de erro na inserção dos dados através do atributo *message*.

As anotações para o relacionamento de tabelas são definidas por *@JoinColumn*, esta contém atributos que indicam o nome da tabela que será referenciada na chave estrangeira (*name*) a coluna de referência da tabela indicada (*referencedColumnName*) se pode ou não ser nula (*nullable*) e o nome da chave estrangeira (*foreignKey*) que é complementada pela anotação da cardinalidade de relação entre as tabelas *@ManyToOne*, muitos para um (1) esta que não precisa de nenhum atributo. Ainda para cardinalidade pode ser usada à anotação *@OneToMany*, um para muitos (0:n), esta não depende da anotação *@JoinColumn*, porém usa alguns atributos para o mapeamento, *mappedBy*= “nome_da_tabela” que define a tabela que será referência, *cascade*= *CascadeType.ALL* podemos definir a forma como serão propagadas as operações em cascata da entidade para suas referências, *orphanRemoval*= *true* faz com que quando apagado apague também suas os dados dependentes deste (filhos) e *fetch*= *FetchType.LAZY* que escolhe

como os dados serão carregados, com a escolha do atributo *LAZY* os dados só serão carregados quando forem solicitados, assim não sobrecarregando o sistema com inúmeras chamadas de carregamento de dados do SGBD.

Dois arquivos são de suma importância para manter a comunicação do banco de dados com o servidor que gerencia o sistema, mantendo assim a persistência e consulta de dados do SGBD, o primeiro deles o arquivo *persistence.xml* é mostrado na Figura 8.

Figura 8 – Arquivo *persistence.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www
  <persistence-unit name="PC-WebPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/pc-projeto-web</jta-data-source>
    <class>br.edu.ifsul.modelo.Curso</class>
    <class>br.edu.ifsul.modelo.Documento</class>
    <class>br.edu.ifsul.modelo.Encontro</class>
    <class>br.edu.ifsul.modelo.Ficha</class>
    <class>br.edu.ifsul.modelo.Permissao</class>
    <class>br.edu.ifsul.modelo.Tipo</class>
    <class>br.edu.ifsul.modelo.Usuario</class>
    <properties>
        <property name="hibernate.hbm2ddl.auto" value="update"/>
        <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
        <property name="hibernate.transaction.jta.platform"
                  value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
        <property name="hibernate.classloading.use_current_tccl_as_parent" value="false"/>
    </properties>
  </persistence-unit>
</persistence>
```

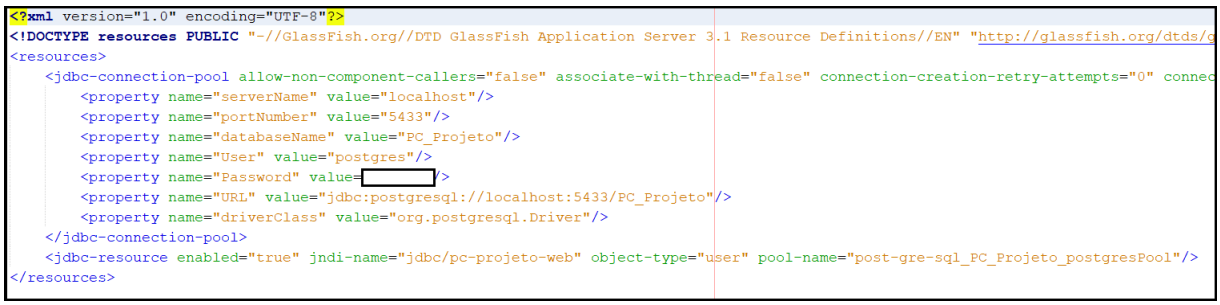
Fonte: do Autor.

Esse arquivo possui os dados referentes à unidade de persistência do sistema e as configurações da comunicação com o banco de dados. Faz parte desse arquivo, o elemento <*persistence-unit*> no qual se escolhe o nome (*name*) da unidade de persistência e o tipo de transição a ser utilizado, no qual foi utilizado o *transaction-type* como Java Transaction API (JTA). O elemento <*provider*> define o Hibernate como provedor de persistência de dados. O elemento <*jta-data-source*> é responsável por indicar o diretório e o nome da fonte de dados em que foi aplicada a unidade de persistência. No elemento <*class*> é definido as classes que fazem parte do projeto. No elemento *properties*, são definidas algumas propriedades da conexão com o banco de dados referentes à consulta e persistência dos dados, a propriedade <*property name= "hibernate.hbm2ddl.auto" value="update"/*> com ela define-se o método automático de atualização das tabelas do banco de dados e a propriedade <*property name= "hibernate.dialect" value=*

“org.hibernate.dialect.PostgreSQLDialect”/> que é responsável pela determinação do “dialeto PostgreSQL” para ser o padrão paras as consultas SQL.

O segundo arquivo é o *glassfish-resources.xml,* mostrado na Figura 9.

Figura 9 – Arquivo *glassfish-resources.xml*



Fonte: do Autor.

Ele é o arquivo responsável para a criação da conexão entre banco de dados e o servidor *GlassFish* que é o servidor responsável pelo controle de instâncias no sistema. O arquivo possui os dados para acesso e autenticação no banco de dados, são elas: *serverName* (Nome do servidor utilizado), *portNumber* (Número da porta de conexão utilizada), *databaseName* (Nome da base de dados utilizada), *User* (Nome do usuário para entrar no sistema), *Password* (Senha do usuário para entrar no sistema), *URL* (Endereço para realizar a conexão) e *driverClass* (Classe responsável pela conexão *JDBC*). O elemento *<jdbc-resource>* contém o nome do diretório da *JNDI* (*Java Naming and Directory Interface*) no atributo *jndi-name*, e o nome do *pool* de conexão ao banco de dados no atributo *pool-name*.

4.2.2 Camada de Controle

A camada de controle é responsável pela comunicação entre os elementos da camada de persistência com a interface gráfica.

Para definir o nome de solicitação das classes de controle, usou-se a anotação *@Named* e seu atributo *value*. Para definir o ciclo de vida do *ManagedBean* das classes de controle da aplicação, utilizou-se a anotação *@ViewScoped*, assim é possível manter os dados durante quantas requisições forem necessárias, desde que todas estas sejam realizadas para a mesma view. Caso seja executado uma requisição para uma pagina e/ou *ManagedBean* diferente, o escopo é limpo, evitando assim que objetos não utilizados se mantenham vivos por muito tempo. Entre as atividades pertinentes das classes de controle

apresentam-se as funções de listar, inserir, editar e excluir dados. A Figura 10 mostra um fragmento do código da classe *ControleFicha*.

Figura 10 – Fragmento de código da classe *ControleFicha*

```
@Named(value = "controleFicha")
@ViewScoped
public class ControleFicha implements Serializable {

    @EJB
    private FichaDAO<Ficha> dao;
    private Ficha objeto;
    private Boolean editando;
    @EJB
    private UsuarioDAO<Usuario> daoUsuario;
    @EJB
    private CursoDAO<Curso> daoCurso;
    @EJB
    private TipoDAO<Tipo> daoTipo;
    private Encontro encontro;
    private Boolean editandoEncontro;
    private Boolean novoEncontro;
    private Documento documento;
    private Boolean editandoDocumento;
    private Boolean novoDocumento;
```

Fonte: do Autor.

A anotação *@EJB* é utilizada para que o servidor fique a cargo do controle de instância das classes *DAO*, assim quando instâncias que não estão mais sendo utilizadas serão descartadas.

4.2.2.1 Classes *DAO*

As classes *DAO (Data Access Object)* da aplicação web desenvolvida, são incumbidas de gerenciar a comunicabilidade entre o banco de dados, a unidade de persistência e os elementos do sistema que fazem uso dos dados persistidos.

O sistema web feito usou uma classe *DAO* genérica, que possui métodos para realizar as operações *CRUD*, filtros e ordenações das buscas realizadas, além de poder definir a quantidade de registros por página e paginação, pode-se visualizar um fragmento do código dessa classe na Figura 11.

Figura 11 – Fragmento de código *DAOGenerico*

```
public class DAOGenerico<TIPO> implements Serializable {

    private List<TIPO> listaObjetos;
    private List<TIPO> listaTodos;
    @PersistenceContext(unitName = "PC-WebPU")
    protected EntityManager em;
    protected Class classePersistente;
    protected String ordem = "id";
    protected String filtro = "";
    protected Integer maximoObjetos = 2;
    protected Integer posicaoAtual = 0;
    protected Integer totalObjetos = 0;
```

Fonte: do Autor.

Para realizar a comunicação com o *SGBD*, as classes *DAO* instanciam um gerenciador de entidades, essa instância se chama *EntityManager*. Com ele pode-se manusear os dados por meio da unidade de persistência, anteriormente criada no arquivo *persistence.xml*, determinada pela anotação *@PersistenceContext*.

No caso das classes *DAO* distintas, elas recebem como parâmetro a classe da camada de modelo a ser persistida. A classe citada possui a função de implementar os métodos oriundos da classe *DAOGenerico*. Pode-se também definir um padrão para ordenação dos resultados de pesquisa. Com o objetivo de atender as especificidades de cada classe do sistema, foram sobrescritos ou implementados os métodos da classe genérica. Por exemplo, vemos o código da classe *FichaDAO* na Figura 12.

Figura 12 – Código da classe *FichaDAO*

```
@Stateful
public class FichaDAO<TIPO> extends DAOGenerico<Ficha> implements Serializable {

    public FichaDAO(){
        super();
        classePersistente = Ficha.class;
        ordem = "tema"; // define a ordem padrão do DAO
        maximoObjetos = 3;
    }

    @Override
    public Ficha getObjectById(Object id) throws Exception {
        Ficha obj = em.find(Ficha.class, id);
        /**
         * A linha obj.getPermissoes().size(); é necessária para inicializar a coleção
         * para quando ela for exibida na tela não gerar um erro de
         * lazyInicializationException
         */
        obj.getDocumentos().size();
        obj.getEncontros().size();
        return obj;
    }
}
```

Fonte: do Autor.

A classe *FichaDAO* recebe a herança de *DAOGenerico* então pode acessar todos os seus métodos, a anotação *@Stateful* é usada para guardar as informações da sessão atual. No exemplo citado, o método *getObjectById* é sobrescrito da classe *DAOGenerico*, para que ele consiga recuperar as informações da lista de documento e encontros presente na classe *Ficha*.

4.2.2.2 Conversores

As páginas web são compostas somente por textos, estes textos são interpretados pelo navegador e exibidos para o usuário. Devido a isso, torna-se necessária a conversão dos dados de quaisquer tipos da aplicação para o tipo *String*, para que seja possível exibi-los na interface com o usuário**.**

Essa conversão se torna possível devido a implementação da interface Converter, disponibilizada pelo pacote *javax.faces*. Por meio dos métodos abstratos implementados pela interface *Converter*, pode-se receber uma *String* por parâmetro e retornar um objeto da classe determinada, através do método *getAsObject()*, no contrário usa-se o método *getAsString()*, o qual converte um objeto em uma *String*. O código do conversor do objeto tipo *Calendar* é apresentado na Figura 13.

Figura 13 – Código da classe *ConverterCalendar*

```
@FacesConverter(value = "converterCalendar")
public class ConverterCalendar implements Serializable, Converter {

    private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

    @Override
    public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
        try {
            Calendar retorno = Calendar.getInstance();
            retorno.setTime(sdf.parse(string));
            return retorno;
        } catch (Exception e){
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext fc, UIComponent uic, Object o) {
        if (o == null){
            return null;
        }
        Calendar obj = (Calendar) o;
        return sdf.format(obj.getTime());
    }
```

Fonte: do Autor.

A anotação *@FacesConverter* é utilizada para definir um nome para a chamada desse conversor posteriormente.

4.2.3 Camada de visão

Essa camada é responsável pela apresentação, é a interface de representação do modelo, ou seja, trata-se da fronteira entre usuário e o sistema em si. A *view* pode dar forma mais conveniente, exibir alguns atributos e ocultar outros, atuando como um filtro para os dados do modelo. É papel da camada de controle definir a *view* apropriada para a exibição da resposta obtida pela requisição feita.

O sistema desenvolvido utilizou o *JSF* em conjunto com as bibliotecas *Primefaces* e *BootsFaces* para elaborar uma interface responsiva.

O arquivo *template.xhtml* foi utilizado como a base do layout da aplicação desenvolvida, ele contém escopos adaptáveis para cada parte de conteúdo, podemos visualizar na Figura 14 a estrutura utilizada para montar o *layout* da aplicação.

Figura 14 – Código do arquivo *template.xhtml*

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:b="http://bootsfaces.net/ui"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title><ui:insert name="titulo"> </ui:insert></title>
    </h:head>
    <h:body>
        <b:container>
            <h:form id="formMenu">
                <b:navBar brand="PC">
                    <b:navbarLinks>
                        <b:navLink icon="home" outcome="#{controleLogin.home()}"/>
                        <b:dropMenu ...11 linhas />
                        <b:dropMenu ...11 linhas />
                    </b:navbarLinks>
                </b:navBar>
            </h:form>
            <ui:insert name="conteudo">

            </ui:insert>
        </b:container>
    </h:body>
</html>
```

Fonte: do Autor.

Os elementos *<ui:insert>* são usados para alterar o conteúdo de uma página mantendo o mesmo layout, assim elas foram utilizadas para modificação do titulo de cada página e também do conteúdo do corpo da página. A aplicação também utiliza o elemento *<h:form>* para manter um menu constante para todas as páginas, esse menu possui o elemento *<b:dropmenu>,* que permite o acesso ao *CRUD* das classes, se o usuário tiver o acesso necessário para tal ação. Nota-se que o menu só é visível se o usuário estiver autenticado e tiver a permissão necessária para acesso as classes, esse controle é feito através do atributo *rendered*, que só permite a visualização do elemento no caso de atingir as condições exigidas serem atingidas.

As páginas de listagens dos registros da aplicação possuem filtros para consultas, ordem de visualização por propriedade do registro, quantidade de itens exibidos por página e também botões para as ações *CRUD*. No caso da lista de Fichas também possui um botão adicional para a impressão do relatório, pode-se visualizar um trecho do código do arquivo *listar.xhtml* da classe Ficha na Figura 15.

Figura 15 – Fragmento de código do arquivo *listar.xhtml* da classe Ficha

```
<p:dataTable value="#{controleFicha.dao.listaObjetos}"
             var="obj" reflow="true" id="listagem">
    <f:facet name="header">
        <div ...21 linhas />
    </f:facet>
    <f:facet ...11 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column ...3 linhas />
    <p:column headerText="Tipo">
        <p:outputLabel value="#{obj.tipo.descricao}"/>
    </p:column>
    <p:column headerText="Ações">
        <div align="center">
            <p:commandButton icon="ui-icon-pencil"
                             actionListener="#{controleFicha.alterar(obj.id)}"
                             process="@this"
                             update="formListagem :formEdicao"/>
            <p:commandButton icon="ui-icon-trash"
                             actionListener="#{controleFicha.excluir(obj.id)}"
                             process="@this"
                             update="formListagem"/>
            <p:commandButton icon="ui-icon-print" action="#{controleFicha.imprimir(obj.id)}"
            ajax="false" onclick="this.form.target= '_blank';"/>
        </div>
    </p:column>
</p:dataTable>
```

Fonte: do Autor.

Para executar as ações citadas anteriormente foi utilizado o elemento *<p:commandButton>* que ao ser acionado executa a função ligada a ele, junto com alguns atributos: *icon,* que define um ícone para o botão, *actionListener,* que define a função a ser chamada, *process,* que determina qual elemento ele acionará e *update*, o qual define os elementos que ele atualizará após a execução da função chamada.

O elemento *<p:dataTable>* é onde são visualizados os dados chamados através da função no elemento *value*, o elemento *<f:facet>* é onde ficam os elementos para ordenação, filtros e definição do número de itens na página e o elemento *<p:column>* é onde os dados de cada coluna do objeto do banco de dados são visualizados.

Para a inclusão de novos registros ou edição dos já cadastrados utilizou-se um arquivo denominado *formulário.xhtml*, podemos visualizar parte de seu código na Figura 16.

Figura 16 – Fragmento de código do arquivo *formulário.xhtml* de Ficha

```
<p:messages id="messages"/>
<p:panelGrid columns="2" columnClasses="ui-grid-col-2, ui-grid-col-20"
    layout="grid" styleClass="ui-panelgrid-blank">
    <f:facet ...3 linhas />
    <p:outputLabel value="ID"/>
    <p:inputText value="#{controleFicha.objeto.id}"
                 size="5" readonly="true"/>
    <p:outputLabel value="Tema" for="txtTema"/>
    <p:inputText ...2 linhas />
    <p:outputLabel value="Semestre" for="txtSemestre"/>
    <p:inputText ...2 linhas />
    <p:outputLabel value="Curso" for="selectCurso"/>
    <p:selectOneMenu value="#{controleFicha.objeto.curso}"
                     id="selectCurso" filter="true" filterMatchMode="startsWith">
        <f:converter converterId="converterCurso"/>
        <f:selectItem itemLabel="Selecione um registro"
                      noSelectionOption="true"/>
        <f:selectItems value="#{controleFicha.daoCurso.listaTodos}"
                       var="c" itemLabel="#{c.descricao}"/>
    </p:selectOneMenu>
    <p:outputLabel value="Tipo" for="selectTipo"/>
    <p:selectOneMenu ...8 linhas />
    <p:outputLabel value="Professor" for="selectProfessor"/>
    <p:selectOneMenu ...8 linhas />
    <p:outputLabel value="Aluno" for="selectAluno"/>
    <p:selectOneMenu ...8 linhas />
    <p:outputLabel value="Coorientador" for="selectCoorientador"/>
    <p:selectOneMenu ...8 linhas />
    <p:commandButton value="Salvar"
                     icon="ui-icon-disk"
                     actionListener="#{controleFicha.salvar()}"
                     update="formEdicao :formListagem"/>
```

Fonte: do Autor.

Neste arquivo é utilizado o elemento *<p:messages>* para exibição das mensagens do sistema, seja de êxito, falha, ou validações em operações e campos. O elemento *<p:panelGrid>* é utilizado para a criação de um formulário de edição de dados onde o *<p:outputLabel>* equivale ao texto de informação e o *<p:inputText>* para a inserção do dado pedido, esse ultimo que é ligado ao atributo do objeto, também temos o elemento *<p:selectOneMenu>* que é usado para relações com outras tabelas do banco de dados, nele são listados todos os registros ligados a chave estrangeira da tabela. Nesse elemento também deve-se usar um conversor similar ao apresentado anteriormente, além disso com ele é possível fazer filtros dos dados para a seleção, e é possível escolher apenas um dado da lista apresentada.

4.2.3.1 Relatório

Para o desenvolvimento do relatório alguns passos foram executados, a começar pela criação da classe *UtilRelatorios*, a Figura 17 apresenta um trecho de seu código.

Figura 17 – Fragmento de código da classe *UtilRelatorios*

```
public static void imprimeRelatorio(String relatorioNome, HashMap parametros, List lista) {
    try {
        JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(lista);
        FacesContext facesContext = FacesContext.getCurrentInstance();
        facesContext.responseComplete();
        ServletContext scontext = (ServletContext) facesContext.getExternalContext().getContext();
        String path = scontext.getRealPath("/WEB-INF/relatorios/");
        parametros.put("SUBREPORT_DIR", path + File.separator);
        JasperPrint jasperPrint
                = JasperFillManager.fillReport(scontext.getRealPath("/WEB-INF/relatorios/" +
                        relatorioNome + ".jasper"), parametros, dataSource);
        byte[] b = JasperExportManager.exportReportToPdf(jasperPrint);
        HttpServletResponse res = (HttpServletResponse)
                FacesContext.getCurrentInstance().getExternalContext().getResponse();
        res.setContentType("application/pdf");
        int codigo = (int) (Math.random() * 1000);
        //Código abaixo gera o relatório e disponibiliza diretamente na página
        res.setHeader("Content-disposition", "inline;filename=relatorio_" + codigo + ".pdf");

        res.getOutputStream().write(b);
        res.getCharacterEncoding();
        FacesContext.getCurrentInstance().responseComplete();
```

Fonte: do Autor.

Este arquivo possibilita a geração do relatório em conjunto com as bibliotecas do *JasperReports*. No método *imprimeRelatorio* é informado para a geração do relatório o diretório de onde estão salvos os *templates* montados no *JasperReports*, o formato em que o mesmo será gerado e as informações que serão impressas – essas que vem através de uma lista com os dados do objeto que é recuperado

através de um *ID* e um *HashMapa* com o mapeamento dos parâmetros, gerados em um método presente no arquivo *ControleFicha*, a Figura 18 mostra o método utilizado na geração de relatórios.

Figura 18 – Método *imprimir(Integer id)* presente na classe *ControleFicha*

```
public void imprimir(Integer id){
    try {
        objeto= dao.getObjectById(id);
    } catch (Exception ex) {
        Logger.getLogger(ControleFicha.class.getName()).log(Level.SEVERE, null, ex);
    }
    List<Ficha> lista= new ArrayList<>();
    lista.add(objeto);
    HashMap parametros= new HashMap();
    UtilRelatorios.imprimeRelatorio("RelatorioFicha", parametros, lista);
}
```

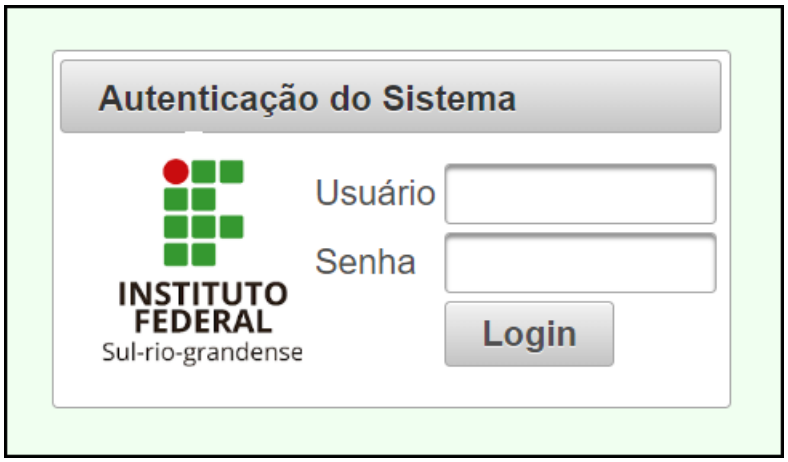
Fonte: do Autor.

5. RESULTADOS

Com o objetivo de aperfeiçoar o processo de gerenciamento dos encontros de orientação, criou-se um sistema web para o auxilio dessa tarefa. Com a conclusão desse sistema tornou-se possível à criação das fichas de gerenciamento das orientações e a impressão do relatório das mesmas. Assim, uma demonstração do processo de criação de fichas, bem como a adição de encontros e a impressão do relatório será descrita nesta sessão.

O sistema conta com uma tela de *login*, na qual o usuário deve estar previamente cadastrado para poder efetuar o acesso ao sistema. A Figura 19 apresenta a tela de *login* do sistema.

Figura 19 – Tela de login do sistema



Fonte: do Autor.

Após a autenticação, o usuário terá acesso ao sistema de acordo com suas permissões. Um usuário com permissão de *ALUNO* poderá visualizar e editar suas informações pessoais, consultar as fichas as quais ele está envolvido, consultar os encontros referentes às orientações já registradas pelo seu professor orientador, adicionar documentos da atual versão do PC e efetuar a impressão do relatório contendo os dados pertinentes da ficha e os encontros registrados, este ultimo no formato disponibilizado pelo documento da instituição. Já um usuário com permissão de *PROFESSOR,* com exceção de adicionar documentos da versão atual do PC – esta é trocada pelo documento da revisão do PC -, poderá fazer as mesmas ações do usuário com permissão *ALUNO*, além de poder criar, editar e excluir novas fichas e encontros. A Figura 20 a seguir mostra a tela inicial do sistema que mostra os dados pessoais do usuário autenticado e as fichas a qual ele está vinculado, as

fichas que são mostradas em um painel que pode ser expandido ou recolhido de acordo com as informações de qual ficha o usuário deseja ver.

Figura 20 – Tela inicial do sistema, *login de usuário* Aluno

Fonte: do Autor.

Para a edição dos dados pessoais, um *Dialog* é aberto após o clique no botão *'Editar Informações'*, a Figura 21 mostra o *Dialog* e o formulário para a edição dos dados.

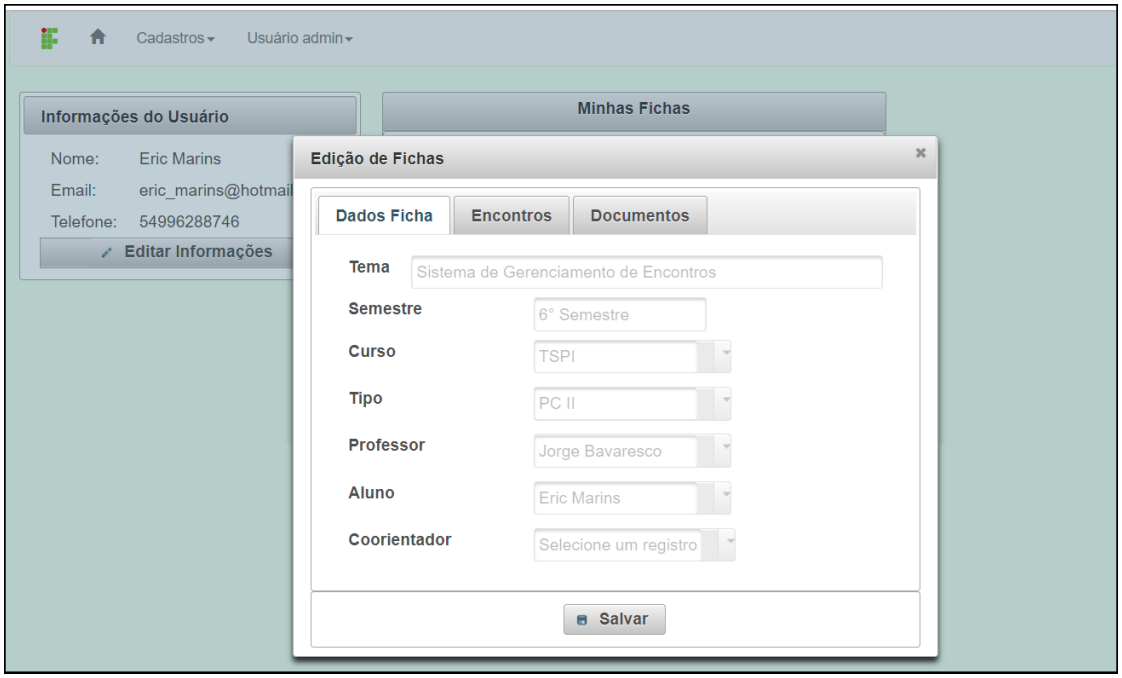
Figura 21 – *Dialog* de edição das informações do usuário

Fonte: do Autor.

O elemento *'Minhas Fichas'*, conta com as fichas relacionadas ao usuário, nesses elementos *ficha* se tem um *menu* com opções para edição, exclusão e a geração do relatório, essas que são mostradas de acordo com a permissão do usuário. A edição de *Fichas*, *Encontros* e *Documentos* é divida através do elemento

Tab em um elemento *Dialog*, a Figura 22 mostra a *Tab* de edição das informações de *Ficha*.

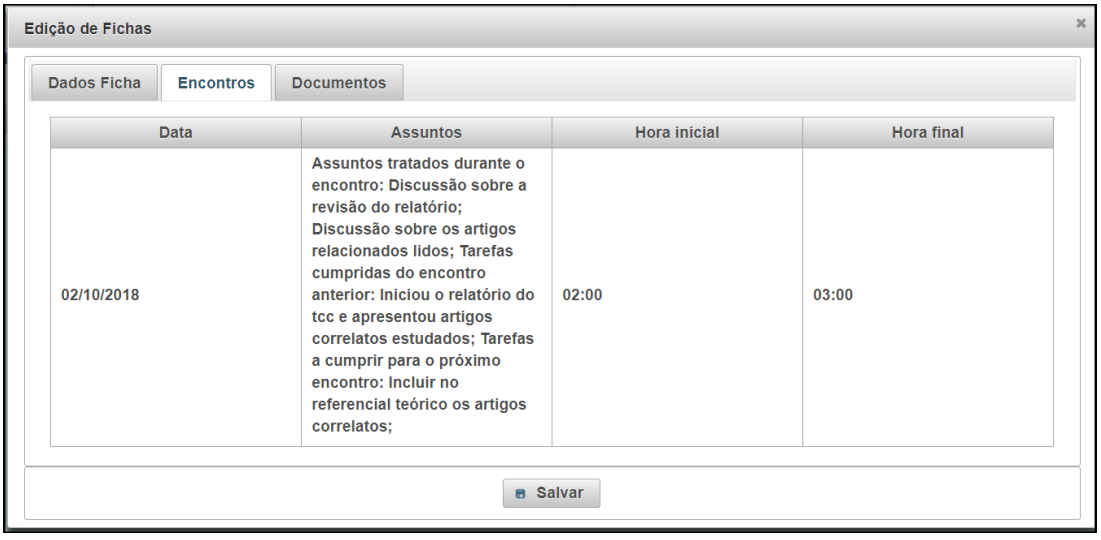
Figura 22 – *Tab* de edição das informações da *Ficha*



Fonte: do Autor.

A Figura 23 mostra a *Tab* de visualização das informações dos *Encontros* já registrados.

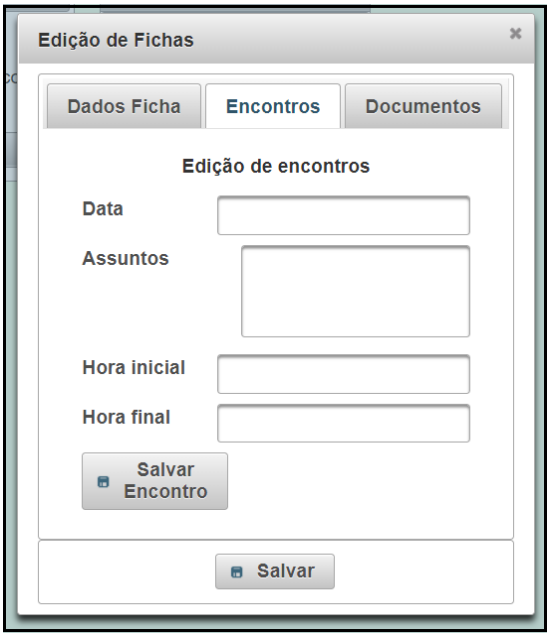
Figura 23 – *Tab* de visualização das informações de *Encontros*



Fonte: do Autor.

Inclusão ou edições dos *Encontros* podem ser feitas na *Tab* de *Encontros,* a Figura 24 mostra a tela de visualização de inclusão de um novo *Encontro*.

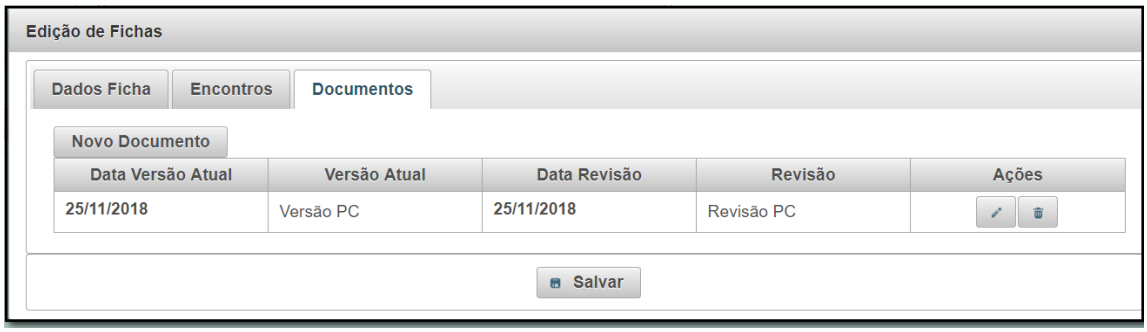
Figura 24 – *Tab* de edição das informações de *Encontro*



Fonte: do Autor.

A *Tab Documentos* mostra as informações dos documentos já enviados, junto com a data do seu envio que é pega automaticamente do sistema no momento do envio do documento. A Figura 25 mostra a visualização das informações da *Tab Documentos.*

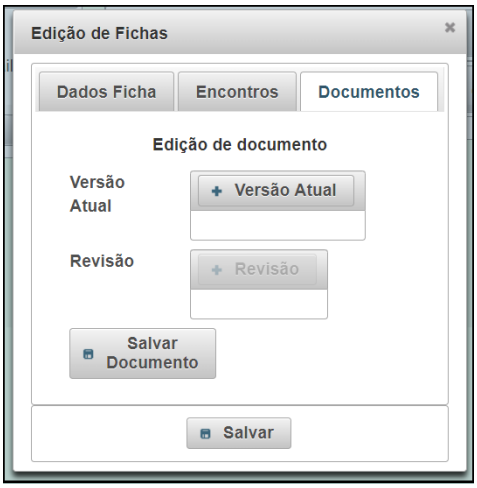
Figura 25 – *Tab* de visualização das informações de *Documentos*



Fonte: do Autor.

A edição da *Tab Documentos* é feita da mesma maneira que a *Tab Encontros*, apenas o envio dos documentos é limitado à permissão do usuário autenticado, apenas usuários Alunos podem enviar a versão do PC, e apenas usuários Professores podem enviar uma revisão. A Figura 26 mostra a tela para a inclusão de um novo *Documento* com um usuário com permissão de Aluno autenticado.

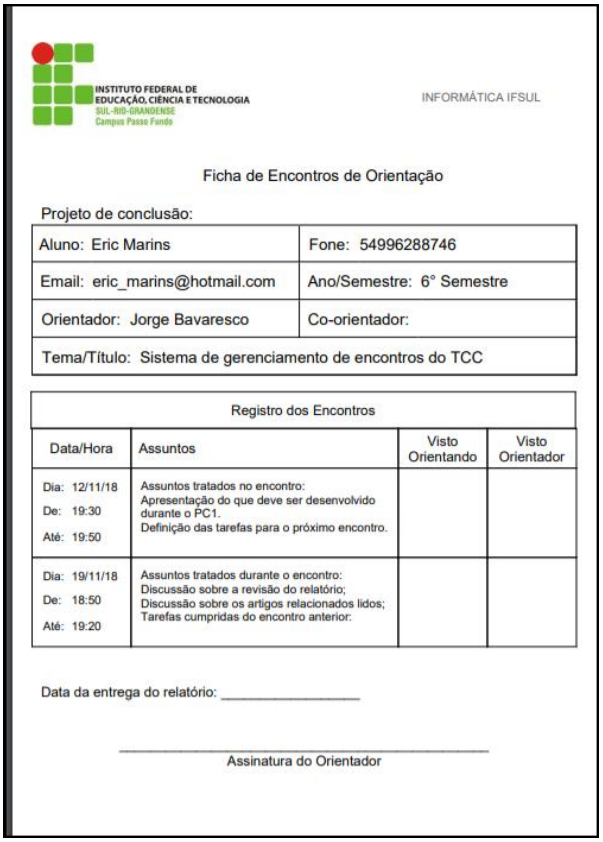
Figura 26 – *Tab* de edição das informações de *Documento*



Fonte: do Autor.

Para a geração do relatório, qualquer usuário autenticado pode gerar sua visualização para impressão, usuários Alunos e Professores podem gerar os relatórios que estão ligados a eles, e um usuário Administrador pode gerar o relatório de qualquer *Ficha* cadastrada. A Figura 27 mostra o relatório gerado a partir de uma Ficha cadastrada.

Figura 27 – Relatório gerado



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE Campus Passo Fundo

INFORMÁTICA IFSUL

Ficha de Encontros de Orientação

Projeto de conclusão:

Aluno: Eric Marins	Fone: 54996288746
Email: eric_marins@hotmail.com	Ano/Semestre: 6° Semestre
Orientador: Jorge Bavaresco	Co-orientador:
Tema/Título: Sistema de gerenciamento de encontros do TCC	

Registro dos Encontros

Data/Hora	Assuntos	Visto Orientando	Visto Orientador
Dia: 12/11/18 De: 19:30 Até: 19:50	Assuntos tratados no encontro: Apresentação do que deve ser desenvolvido durante o PC1. Definição das tarefas para o próximo encontro.		
Dia: 19/11/18 De: 18:50 Até: 19:20	Assuntos tratados durante o encontro: Discussão sobre a revisão do relatório; Discussão sobre os artigos relacionados lidos; Tarefas cumpridas do encontro anterior:		

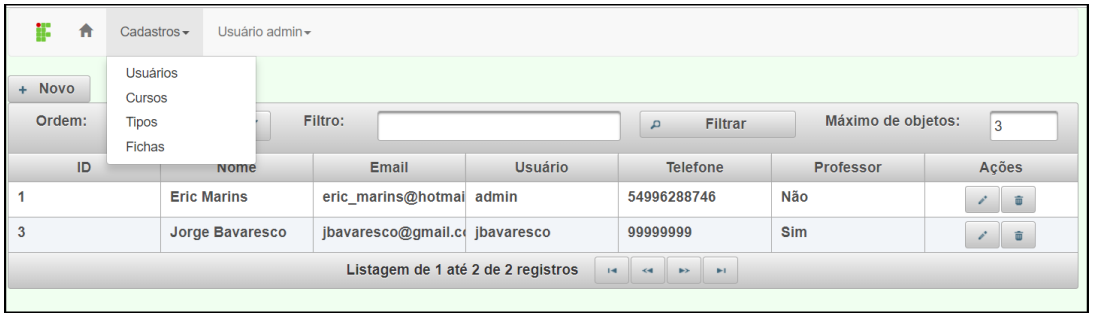
Data da entrega do relatório: _____

_____
Assinatura do Orientador

Fonte: do Autor.

O sistema ainda conta com a permissão *ADMINISTRADOR*, que permite ao usuário acessar um *menu* de acesso ao *CRUD* de todas as classes, permitindo-o assim cadastrar novos usuários, além de outras classes. A Figura 28 mostra o *menu* disponível aos usuários com a permissão de *ADMINISTRADOR* e também mostra a lista dos usuários cadastrados no sistema.

Figura 28 – *Menu* de acesso aos *CRUDs* e lista de usuários cadastrados



Fonte: do Autor.

Assim como levantado como requisito anteriormente, os elementos do sistema foram desenvolvidos de forma responsiva, foram feitos testes por meio do emulador do navegador Google Chrome no modo *Device* no formato do celular *'Galaxy S5'*, a Figura 29 mostra o resultado destes testes.

Figura 29 – Teste de design responsivo do sistema

Fonte: do Autor.

6. CONSIDERAÇÕES FINAIS

O presente estudo teve por objetivo desenvolver um sistema para o gerenciamento dos encontros de orientação do Trabalho de Conclusão de Curso, no qual sua finalidade é de padronizar a documentação dessa atividade, que é obrigatória para a defesa perante a banca do TCC, e tornar mais fácil o acesso às informações discutidas em orientações prévias, resultando em um acompanhamento melhor tanto de orientador como de orientando.

O sistema apresentado teve seu levantamento de requisitos através do documento disponibilizado pela instituição, após foi feita a modelagem de dados em cima do estudo de caso, e por fim o desenvolvimento da aplicação web, utilizando os recursos necessários para que o sistema englobasse todas as funções levantadas no estudo de caso.

Em vista disso, o desenvolvimento do sistema foi feito através da integração de diversas tecnologias, sendo algumas: a linguagem de programação Java, a plataforma *Java EE*, que disponibiliza recursos como os *containers EJB* que em conjunto com o servidor *GlassFish* controlam as instâncias de objetos da aplicação, o *JSF* em conjunto das bibliotecas *PRIMEFACES* e *BOOTSFACES* para a criação da interface gráfica responsiva, o conjunto de *JPA* e *Hibernate* para a validação e persistência dos dados no banco de dados relacional o qual foi utilizado o *SGBD PostgreSQL*. Concluindo, as funcionalidades propostas no presente trabalho foram implementadas com sucesso.

No decorrer do trabalho a maior dificuldade encontrada foi a disponibilidade de tempo para a realização do trabalho em meio à rotina diária. Para trabalhos futuros pode ser pensadas e implementadas novas funcionalidades para o sistema, tal como um acervo com os TCCs defendidos.

REFERÊNCIAS

BootsFaces. **BootsFaces Showcase and Documentation**. Disponível em< https://showcase.bootsfaces.net/> Acesso em: 05 mai. 2018.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. São Paulo: Pearson Prentice Hall, 2010.

GEARY, David; HORSTMANN, Cay. **Core JavaServer™ Faces**. Rio de Janeiro: ALTA BOOKS, 2012.

GONÇALVES, Antonio. **Beginning Java EE 7.** Apress, 2013.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, EJB3 Persistence e Ajax**. Rio de Janeiro: Ciência Moderna, 2007.

HEFFELFINGER, David R.. **JasperReports 3.5 for Java Developers**. Packt Publishing, 2009.

JUNIOR, Joelson; CARVALHO, Cíntia; BORIM, Andrea. **FERRAMENTA MOODLE COMO RECURSO PARA GERENCIAMENTO E ORIENTAÇÃO DE TCC**. Anuário da Produção Acadêmica Docente - Faculdade de Negócios e Tecnologia da Informação, Vol. 5, N°. 10, 2011. Disponível em: <http://repositorio.pgsskroton.com.br/bitstream/123456789/1430/1/Artigo%203.pdf>. Acesso em: 11 abr. 2018.

MASO, Lucas. **MODELAGEM DE UM SISTEMA DE INFORMAÇÃO APLICADO AO GERENCIAMENTO DO ACOMPANHAMENTO DE TRABALHOS DE CONCLUSÃO DE CURSO DO IFC-CAMPUS CAMBORIÚ.** 2016. Disponível em: <https://bit.ly/2l8DuWC>. Acesso em: 11 abr. 2018.

MILANI, André. **PostgreSQL: guia do programador**. São Paulo: NOVATEC EDITORA LTDA, 2008.

OLIVEIRA, Rodrigo. **SISTEMA DE GERENCIAMENTO DE TCCS DO CURSO DE SISTEMAS DE INFORMAÇÃO DA UNIPLAC**. 2010. Disponível em:

<https://revista.uniplac.net/ojs/index.php/tc_si/article/viewFile/879/589>. Acesso em: 11 abr. 2018.

Oracle. **About the Java Technology**. Disponível em: < https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>. Acesso em: 11 abr. 2018.

PRIMEFACES. **Why PrimeFaces**. Disponível em < http://www.primefaces.org/whyprimefaces>. Acesso em 05 mai. 2018.

ROMAN, Ed; AMBLER, Scott; JEWELL, Tyler. **Dominando Enterprise Javabeans**. Porto Alegre: ARTMED® EDITORA S.A, 2008.

SAMPAIO, Cleuton. **Java Enterprise Edition 6-Desenvolvendo Aplicações Corporativas**. Brasport, 2011.

ANEXO A – Documento disponibilizado pela instituição para entrega do registro dos encontros de orientação do TCC

INFORMÁTICA IF SUL

Ficha de Encontros de Orientação (Anexo 1)

Projeto de Conclusão:

Aluno:	**Fone:**
E-mail:	**Ano/Semestre:**
Orientador:	**Co-orientador:**
Tema/Título:	

Registros dos encontros			
Data/Hora	**Assuntos**	**Visto Orientando**	**Visto Orientador**

Data de entrega do relatório: _____

_____
Assinatura do Orientador