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

MARLON EDUARDO BERTAN

SISTEMA PARA GERENCIAMENTO DE ATENDIMENTOS DOS SETORES NAPNE/NUAPE DA UTFPR

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

MARLON EDUARDO BERTAN

SISTEMA PARA GERENCIAMENTO DE ATENDIMENTOS DOS SETORES NAPNE/NUAPE DA UTFPR

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

Orientador: Vinícius Pegorini

PATO BRANCO
2015

SISTEMA PARA GERENCIAMENTO DE ATENDIEMNTOS DOS SETORES NAPNE/NUAPE DA UTFPR

Por

Marlon Eduardo Bertan

Esta monografia foi apresentada às 16h30 do dia 03 de novembro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

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

Profa. Msc. Lucilia Yoshie Araki
Banca
UTFPR – Campus Pato Branco

Profa. Dr. Beatriz Borsoi
Banca
UTFPR – Campus Pato Branco

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

RESUMO

BERTAN, Marlon Eduardo. Sistema para gerenciamento de atendimento de setores NAPNE/NUAPE da UTFPR. 2015. 50 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2015.

O uso de sistemas computacionais para o auxílio na realização de processos nos dias atuais é a cada dia mais evidente. Isso se deve a grande quantidade de informações geradas e manipuladas. Devido a isso, não podemos deixar pessoas armazenarem informações importantes apenas em agendas e cadernos, especialmente quando estamos tratando de áreas relacionadas a saúde. Ter as informações do paciente a mão rapidamente, tem uma importância muito grande para que a tomada de decisão do profissional que está atendendo seja a mais correta possível para determinada situação, não ocasionando atrasos e/ou enganos, trazendo eficiência e eficácia ao atendimento/agendamento realizado. O sistema de agendamento e atendimento é um sistema *web* implementado na linguagem Java com o auxílio de vários *frameworks*, possibilitando o agendamento de consultas e atendimento das mesmas, assim como o cadastramento de quem irá atender e quem será atendido.

Palavras-chave: Java para web. Aplicações Internet Ricas. Sistema de agendamento.

ABSTRACT

BERTAN, Marlon Eduardo. Software to manager treatment in the UTFPR NAPNE/NUAPE sectors. 2015. 50 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2015.

The use of computer systems for assistance in carrying out processes in the present day is more evident every day. This is due to the large amount of information generated and manipulated. Because of this, we cannot let people store important information only in diaries and notebooks, especially when we are dealing with areas related to health. Have the information of the patient's hand quickly, has great importance for the professional decision-making that is serving is the truest possible for a given situation, not causing delays and/or mistakes, bringing efficiency and effectiveness of care/scheduling accomplished. The Scheduling and treatment system is a web system implemented in the Java language with the help of various frameworks, allowing for scheduling appointments and answering the same, as well as the registration of who will attend and who will be attending.

Keywords: Java web. Rich Internet Application. Scheduling System.

LISTA DE FIGURAS

Figura 1 – Diagrama de Casos de Uso 20
Figura 2 - Diagrama de Entidade e Relacionamento 21
Figura 3 – Tela de Login da Aplicação 22
Figura 4 - Tela Inicial do Sistema 23
Figura 5 - Módulo de Cadastros 24
Figura 6 - Listagem de Departamento(s)/Setor(es) 24
Figura 7 – Validação Departamento/Setor 24
Figura 8 - Cadastro de Departamento/Setor 25
Figura 9 - Listagem de Especialidades 25
Figura 10 – Mensagem de Validação ao excluir um registro 26
Figura 11 - Cadastro de Especialidade 26
Figura 12 - Listagem de Pacientes 27
Figura 13 - Cadastro de Paciente 27
Figura 14 - Listagem de Usuários 27
Figura 15 - Cadastro de Usuários 28
Figura 16 - Módulo Agendamento – Menu Realizar Agendamento 29
Figura 17 - Listagem de Agendamentos 29
Figura 18 - Manutenção e Inserção de Agendamentos 30
Figura 19 - Módulo Atendimento – Menu Realizar Atendimento 30
Figura 20 - Listagem de Atendimentos 30
Figura 21 – Geração de um novo atendimento 31
Figura 22 – Módulo Listagem – Menu Listagem de Atendimento 31
Figura 23 – Pesquisa de Atendimentos 32
Figura 24 – Listagem de Atendimentos 32
Figura 25 – Consulta de um atendimento 33

LISTA DE QUADROS

Quadro 1 - Comparação entre desktop, web tradicional e RIA 14
Quadro 2 – Ferramentas e Tecnologias utilizadas 15
Quadro 3 – Listagem dos requisitos funcionais 19

LISTAGENS DE CÓDIGOS

Listagem 1 – Template Padrão **34**
Listagem 2 – Layout Genérico das Páginas **34**
Listagem 3 - Criação dos botões Novo, Editar e Excluir **35**
Listagem 4 - Criação da Listagem de Pacientes **35**
Listagem 5 - JavaScript responsável pela seleção de uma linha da tabela e validações **36**
Listagem 6 - Campo Nome, do cadastro de Pacientes **37**
Listagem 7 - Campo Departamento, do cadastro de Pacientes **37**
Listagem 8 – Classe Conversora de Departamentos **38**
Listagem 9 - Criação dos botões Salvar e Cancelar do Cadastro de Pacientes **38**
Listagem 10 - Método Cancelar do Bean do Paciente **38**
Listagem 11 - Método Salvar do Bean do Paciente **39**
Listagem 12 - Método Editar do Bean de Paciente **39**
Listagem 13 – Método Excluir do Bean de Paciente **39**
Listagem 14 – Utilizando Injeção de Dependência com CDI **40**
Listagem 15 – Método Inicializar do Bean do Paciente **40**
Listagem 16 – Entidade do Paciente – Anotações de Tabela **40**
Listagem 17 – Entidade do Paciente – Anotações de identificador da tabela **41**
Listagem 18 – Entidade do Paciente – Junção entre Tabelas **41**
Listagem 19 – Interface GenericDAO **41**
Listagem 20 – Classe GenericDAOImpl **43**
Listagem 21 – Interface PacienteDAO **43**
Listagem 22 – Classe PacienteDAOImpl **44**
Listagem 23 – Interface GenericController **44**
Listagem 24 – Interface PacienteController **44**
Listagem 25 - Classe PacienteControllerImp **45**
Listagem 26 - Pom.xml, informações do projeto **46**
Listagem 27 – Pom.xml, dependências do projeto **47**

LISTA DE SIGLAS

CDI	*Contexts and Dependency Injection*
CRUD	*Create, Read, Update, Delete*
EJB	*Enterprise Java Beans*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
Java EE	*Java Enterprise Edition*
Java SE	*Java Standard Edition*
JPA	*Java Persistence API*
JSF	*Java Server Faces*
NUAPE	Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil
NAPNE	Núcleo de Atendimento à Pessoa com Necessidades Educacionais Especiais
ORM	*Object-Relational Mapping*
POJO	*Plain Old Java Objects*
RIA	*Rich Internet Applications*
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO 10
1.1 CONSIDERAÇÕES INICIAIS 10
1.2 OBJETIVOS 11
1.2.1 Objetivo Geral 11
1.2.2 Objetivos Específicos 11
1.3 JUSTIFICATIVA 12
1.4 ESTRUTURA DO TRABALHO 12
2 REFERENCIAL TEÓRICO 13
2.1 RICH INTERNET APPLICATIONS 13
3 MATERIAIS E MÉTODO 15
3.1 MATERIAIS 15
3.2 MÉTODO 16
4 RESULTADOS 18
4.1 ESCOPO DO SISTEMA 18
4.3 APRESENTAÇÕES DO SISTEMA 22
4.4 IMPLEMENTAÇÃO DO SISTEMA 32
5 CONCLUSÃO 49
REFERÊNCIAS 50

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho, com as considerações inicias, os objetivos e a justificativa. As considerações iniciais inserem o trabalho proposto no contexto da Universidade e no referencial teórico. Os objetivos explicitam as funcionalidades essenciais do sistema. E a justificativa ressalta a importância e a necessidade do seu desenvolvimento e da principal tecnologia adotada para a implementação do sistema. O capítulo é finalizado com a apresentação dos capítulos subsequentes que compõem o texto.

1.1 CONSIDERAÇÕES INICIAIS

A Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Pato Branco, possui dois setores de atendimento e acompanhamento docente, discente e de técnicos administrativos que estão vinculados ao Departamento de Educação. Esses setores são o Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil (NUAPE) e o Núcleo de Atendimento à Pessoa com Necessidades Educacionais Especiais (NAPNE)

Nesses dois setores, que atualmente compartilham o mesmo espaço físico, trabalham profissionais da área de psicologia e pedagogia, além de técnicos administrativos. As atividades realizadas são basicamente de atendimento e acompanhamento psicológico e pedagógico de alunos, professores e funcionários. Esses atendimentos são realizados pelos profissionais de acordo com as suas habilidades em decorrência da ocorrência de necessidade de atendimento.

Os atendimentos são, em geral, realizados de acordo com a agenda de cada profissional. Contudo, em um mesmo caso, quando há necessidade de acompanhamento por um período de tempo e que pode ser multidisciplinar, os atendimentos podem ser realizados por profissionais distintos dentro da mesma especialidade ou em especialidades distintas. Isso faz com que as agendas sejam exclusivas de cada profissional, mas o registro do atendimento por pessoa (seja docente, discente, técnico administrativo ou outros profissionais que exerçam suas atividades na Universidade) é compartilhado entre os diversos profissionais. Esse compartilhamento se faz necessário, inclusive, pela demanda atual e o profissional

de determinada especialidade que está fazendo o atendimento pode não estar disponível para prosseguir com atendimentos posteriores de um caso já iniciado por ele.

Visando propor um aplicativo para auxiliar no controle de agenda e dos atendimentos realizados pelos profissionais dos setores NAPNE e NUAPE, verificou-se a possibilidade de desenvolver um sistema web que permita o acesso exclusivo de agenda de cada profissional e o compartilhamento entre todos os profissionais dos dados dos atendimentos realizados.

1.2 OBJETIVOS

A seguir estão o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

- Implementar um sistema web para o controle de agenda e de atendimentos realizados pelos profissionais dos setores NAPNE e NUAPE da UTFPR, Câmpus Pato Branco.

1.2.2 Objetivos Específicos

- Fornecer um controle de agenda para os profissionais da área de psicologia e pedagogia da UTFPR.
- Agilizar a localização de informações de atendimentos já realizados.
- Facilitar o acompanhamento de atendimentos realizados e agilizar a interação do profissional com cada caso sendo acompanhando quando os atendimentos realizados são compartilhados por profissionais distintos.

1.3 JUSTIFICATIVA

O desenvolvimento de uma Aplicação Internet Rica (RIA) foi escolhido pelos recursos de interação que a mesma oferece, visando, assim, facilitar o uso por profissionais com formações distintas.

O aplicativo permitirá o compartilhamento dos registros dos atendimentos realizados pelos diferentes profissionais e oferecerá um controle individual de agenda. Essas são as funcionalidades essenciais que justificam o desenvolvimento do aplicativo. Esse tipo de registro tem sido realizado em formulários impressos, dificultando a localização e mesmo o registro dos dados. Acredita-se que um aplicativo para *web* (compartilhamento de dados) facilitará o acesso, agilizando o trabalho de busca e de registro.

A obtenção de dados de atendimentos será facilitada por meio do uso do sistema. Com os dados armazenados futuramente poderão ser implementadas filtragens e agrupamentos de dados de acordo com os interesses e necessidades dos usuários do sistema.

1.4 ESTRUTURA DO TRABALHO

Este texto está estruturado em capítulos. O capítulo 2 apresenta o referencial teórico do trabalho que está centrado em RIA . O Capítulo 3 apresenta os materiais e o método utilizados na modelagem e na implementação do trabalho. O resultado da realização do trabalho é apresentado no Capítulo 4. As considerações finais estão no Capítulo 5. Por fim estão as referências bibliográficas utilizadas na fundamentação do referencial teórico e do método utilizado na modelagem e desenvolvimento do sistema.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho que se refere às aplicações Internet denominadas como ricas.

2.1 RICH INTERNET APPLICATIONS

O termo *Rich Internet Applications* (RIA) que significa aplicação Internet rica foi inserido em um artigo da Macromedia por Jeremy Allaire, datado de março de 2002, para denotar a unificação das aplicações *desktop* tradicionais com as aplicações *web*, visando combinar as vantagens e superar as desvantagens de ambas as tecnologias (BOZZON *et al*., 2006). As RIA combinam áudio, vídeo e troca de mensagens em tempo real com tecnologias de comunicação que tornam a experiência de interação do usuário com as aplicações *web* bastante rica e interessante (PANG *et al*., 2010).

As abordagens RIA possuem tipicamente quatro características chave (STEARN, 2007, p. 67):

a) Ambiente de execução – o desenvolvedor escolhe o aplicativo, denominado navegador *web*, para executar a aplicação e prover suas funcionalidades básicas;
b) Interface gráfica com o usuário – a linguagem selecionada para implementar o projeto de interface é escolhida pelo desenvolvedor e atualmente existem muitas opções de tecnologia para isso;
c) Lógica de negócio – a linguagem selecionada para manipular a interação com o usuário e controlar a interação com o usuário;
d) Serviços de *back-end* – os *links* com outros conteúdos externos à aplicação (qualquer serviço externo que provê acesso a bases de dados, armazenamento de arquivo local ou remoto e aos meios para consumir e agregar recursos baseados em *web*).

O Quadro 1 apresenta uma comparação de características das aplicações *desktop* (incluindo cliente-servidor), das aplicações *web* convencionais (baseadas

em *HyperText Markup Language* (HTML) e *Hypertext Transfer Protocol* (HTTP) com as RIA, que são caracterizadas por um modelo distribuído, interface melhorada em termos de recursos de interação com o usuário e com redução da sobrecarga de comunicação (BOZZON et al., 2006).

Característica	**Desktop, Cliente-Servidor**	**Web tradicional**	**RIA**
Cliente universal (navegador)	Não	Sim	Sim
Recursos de interação	Ricos	Limitados	Ricos
Instalação no cliente	Complexa	Simples	Simples
Lógica de negócios no lado servidor	Sim	Sim	Sim
Lógica de negócios no lado cliente	Sim	Limitada	Sim
Necessidade de atualização da página completa	Não	Sim	Sim
Comunicação frequente com o servidor	Não	Sim	Não
Comunicação do cliente para o servidor	Sim	Não	Sim
Funcionamento sem conexão	Sim	Não	Sim

Quadro 1 - Comparação entre desktop, web tradicional e RIA
Fonte: Bozzon et al. (2006, p. 354).

As RIA são tipicamente carregadas pelo cliente juntamente com os dados iniciais necessários à sua execução, em seguida a aplicação gerencia a atualização dos dados e o processamento de eventos, comunicando com o servidor somente quando o usuário solicita mais informações ou dados devem ser submetidos ao servidor para processamento (BOZZON *et al*., 2006).

Para Linaje, Preciado e Sánchez-Figueroa (2005) as RIA combinam os benefícios do modelo de distribuição que é característica da *web* com a alta interatividade multimídia oferecida pelas aplicações *desktop*.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados na modelagem e na implementação do sistema.

3.1 MATERIAIS

Os materiais se referem às ferramentas, tecnologias, ambientes de desenvolvimento e outros que são utilizados para realizar as atividades desde a definição dos requisitos à implantação do sistema.

No Quadro 2 estão as ferramentas e as tecnologias utilizadas para modelagem e a implementação do sistema.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Eclipse Luna	4.4.0	http://eclipse.org	IDE de desenvolvimento.
JQuery	2.1.1	http://jquery.com	Biblioteca de JavaScript.
DBDesigner	2.25	http://www.fabforce.net/dbdesigner4/	Modelagem do banco de dados.
Linguagem Java	5.6	http://www.oracle.com/java	Linguagem de programação.
Sublime Text	3	http://www.sublimetext.com/	Ambiente de desenvolvimento.
PostgreSQL	9.4.4	http://www.postgresql.org/	Banco de dados.
pgAdmin	1.20.0	http://www.pgadmin.org/	Administrador do banco de dados.
Maven	3.1	http://maven.apache.org/	Gerenciamento de Dependências
Wildfly	8.1	http://wildfly.org/	Servidor *web* para a aplicação.
Bootstrap	3.2.0	http://getbootstrap.com/	Framework para desenvolvimento da interface responsiva.

Quadro 2 – Ferramentas e Tecnologias utilizadas

3.1.1 Java Server Faces (JSF)

É uma especificação Java para a construção de interfaces de usuário baseadas em componentes para aplicações *web*. Possui um modelo de

programação dirigido a eventos e na atual versão utiliza Facelets como seu sistema de *template* padrão.

Ganhou expressão na versão 1.1 e hoje na versão 2.0 é considerado pela comunidade Java como a melhor especificação em termos de desenvolvimento de aplicações Web utilizando Java.

3.1.2 Java Persistence API (JPA)

É uma API padrão da linguagem Java que descreve uma interface comum para *frameworks* de persistência de dados (CORDEIRO, 2012). A JPA define um meio de mapeamento objeto-relacional para objetos Java simples e comuns, os *Plain Old Java Objects* (POJO), denominados *beans* de entidade. Diversos *frameworks* de mapeamento objeto/relacional como o Hibernate implementam a JPA. Também gerencia o desenvolvimento de entidades do Modelo Relacional usando a plataforma nativa *Java Standard Edition* (Java SE) e *Java Enterprise Edition* (Java EE).

3.2 MÉTODO

A seguir estão as fases ou etapas que agrupam as principais atividades realizadas para o levantamento dos requisitos, a modelagem e a implementação do aplicativo. Essas etapas são definidas tendo como base o proposto pelo modelo sequencial linear definido em Pressman (2006). Os procedimentos tiveram esse método como base, embora o aplicativo não tenha sido desenvolvimento completamente em uma única iteração. A escolha do método sequencial linear foi decorrente da simplicidade do sistema em termos de requisitos, basicamente cadastros, controle de agenda e registro de atendimentos, embora alguns requisitos tiveram implementação relativamente complexa.

a) Levantamento de requisitos

O levantamento dos requisitos foi realizado a partir de conversa com profissionais dos núcleos NAPNE e NUAPE da UTFPR, Câmpus Pato Branco. Esses profissionais explicaram a rotina das suas atividades, a forma de encaminhamento de cada tipo de atendimento realizado (para alguns dos atendimentos é solicitado

auxílio de profissionais externos, como o núcleo de psicologia vinculado a uma Faculdade do município) e as atividades de cada profissional. Nessa conversa ficou evidente a necessidade de uma agenda para cada profissional e de possibilidade de compartilhamento dos atendimentos realizados, inclusive, pelo elevado número de atendimentos que têm sido realizados.

Evidenciou-se, assim, a necessidade de permitir o acesso de cada usuário a sua própria agenda para inclusão, mas o compartilhamento dos cadastros e registros e consulta nas agendas dos demais profissionais.

b) Análise e projeto

A partir das definições obtidas com o levantamento de requisitos, uma visão geral do sistema foi elaborada. Essa visão foi registrada através do diagrama de entidades e relacionamentos do banco de dados.

c) Implementação do sistema

O sistema foi implementado utilizando as tecnologias Java EE, como *Enterprise Java Beans* (EJB), utilizada para encapsular a lógica de negócios da aplicação e *Java Persistence API* (JPA), que permite gerenciar os dados utilizando mapeamento relacional de objetos (*Object-Relational Mapping* (ORM)). Assim como sua parte Web, com o framework *Java Server Faces* (JSF), que é uma especificação baseada em componentes para a construção de interfaces de usuário. Outras tecnologias utilizadas foram o Ajax e JavaScript, que são utilizados principalmente do lado cliente da aplicação, permitindo dentre várias coisas, que sejam feitas alterações na página sem a necessidade de atualizá-la.

d) Realização dos testes

Os testes foram informais visando identificar erros de código e foram realizados pelo autor do trabalho.

e) Implantação do sistema

O sistema será implantado em um servidor da própria UTFPR.

4 RESULTADOS

Este capítulo apresenta o que foi obtido como resultado do desenvolvimento do trabalho. Esses resultados estão relacionados à modelagem e a implementação do sistema.

4.1 ESCOPO DO SISTEMA

O sistema de agendamento de consultas foi desenvolvido pensando em facilitar o trabalho diário dos profissionais dos setores NAPNE/NUAPE da UTFPR. Sendo um sistema de simples utilização e aprendizado fácil, contém cadastros de pacientes e profissionais que são respectivamente quem é atendido e quem atende, bem como cadastro de departamentos/setores e especialidades.

O sistema desenvolvido tem como foco principal o agendamento de consultas e o atendimento das mesmas, gerando e armazenando um prontuário para que os registros do paciente possam ser consultados futuramente por meio de relatórios.

4.2 MODELAGEM DO SISTEMA

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

Identificação	Nome	Descrição
01	Realizar Agendamento	O usuário responsável faz o cadastro dos agendamentos no sistema.
02	Realizar Atendimento	O Psicólogo ou Pedagogo responsável pelo atendimento realiza o cadastro do mesmo no sistema.
03	Cadastrar Paciente	O usuário responsável faz o cadastro do Paciente
04	Cadastrar Profissional	O usuário responsável faz o cadastro do Profissional
05	Cadastrar Especialidade	O usuário responsável faz o cadastro da Especialidade
06	Cadastrar Departamento	O usuário responsável faz o cadastro do Departamento
07	Cadastrar Tipo	O usuário responsável faz o cadastro do Tipo de

	de Paciente	Paciente
08	Cadastrar Usuário	O administrador do sistema faz o cadastro do Usuário

Quadro 3 – Listagem dos requisitos funcionais

O cadastro de agendamento e atendimento são as principais funcionalidades do sistema. No cadastro de agendamento o usuário deve informar o paciente, o profissional que o irá atender, a data e a hora do atendimento e ao realizar o atendimento deverá informar o prontuário do paciente.

O diagrama de casos de uso apresentado na Figura 1 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: Usuário Comum, Administrador, Paciente e Profissional.

a) Usuário Comum – o usuário comum é responsável pelo agendamento de consultas e o cancelamento de consultas, quando solicitado.

b) Usuário Administrador – o Usuário Administrador tem apenas a função de cadastrar um novo Usuário Comum.

c) Profissional – o Profissional é responsável por realizar o atendimento e informar o prontuário.

d) O Paciente pode solicitar uma consulta e/ou solicitar cancelamento de um agendamento.

A Figura 1 apresenta o diagrama de casos de uso.

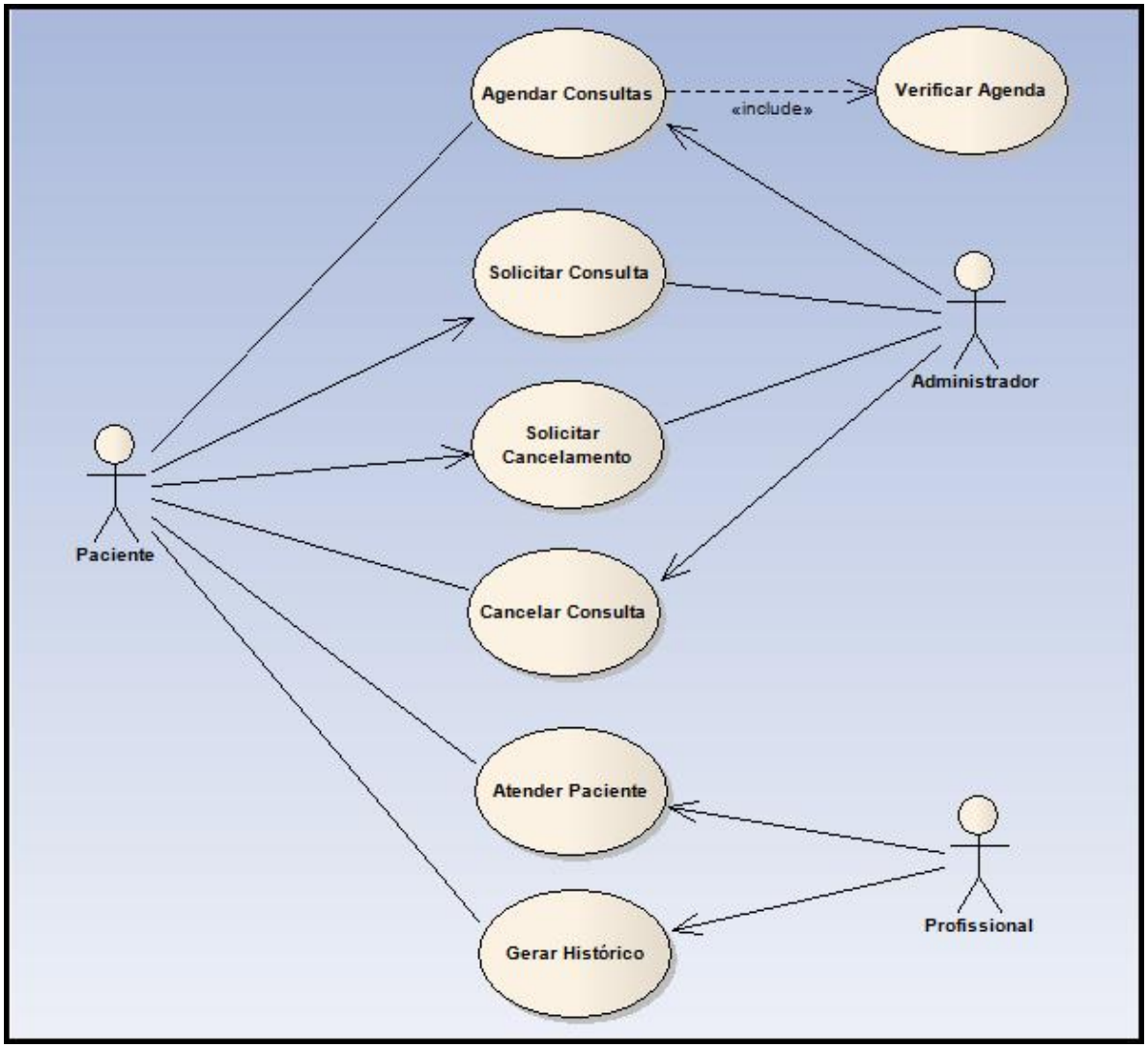

Figura 1 – Diagrama de Casos de Uso

Na Figura 2 é apresentado o digrama de entidade relacionamento, o qual exibe as tabelas do sistema nas quais serão armazenados os dados informados nas telas da aplicação. Em geral, este modelo representa de forma abstrata a estrutura que possuirá o banco de dados da aplicação.

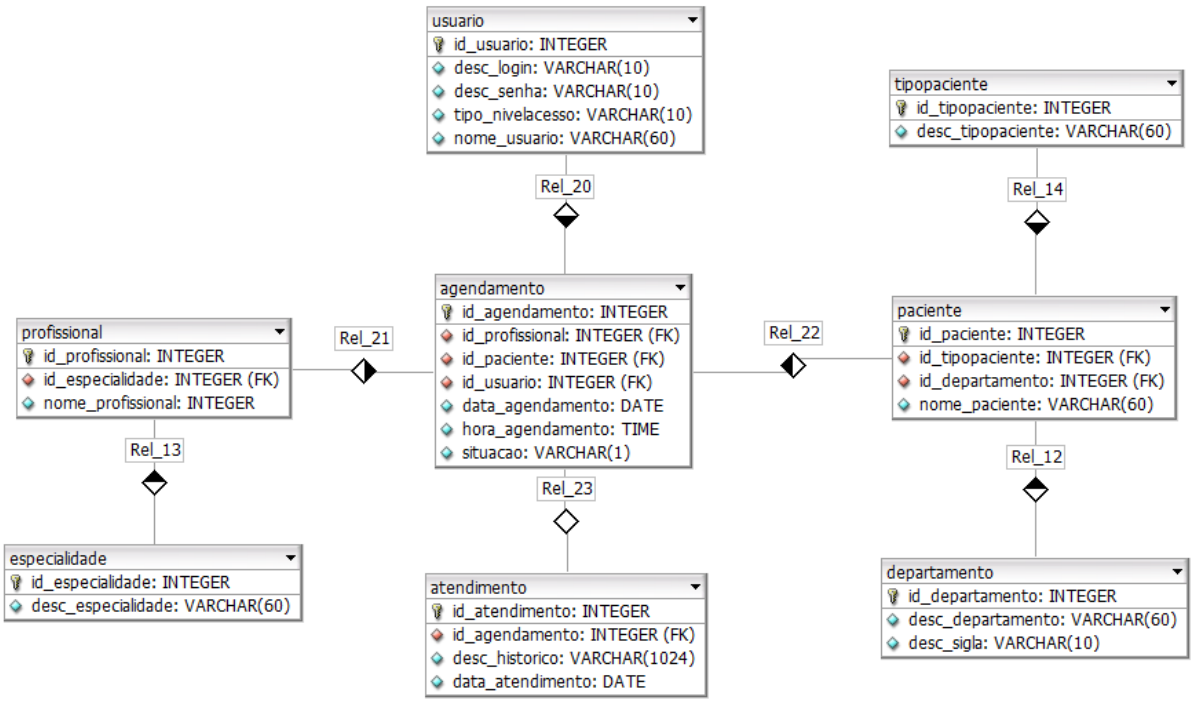

Figura 2 - Diagrama de Entidade e Relacionamento

O sistema possui em um módulo principal de Agendamento que tem um relacionamento de um para muitos com as tabelas usuário, profissional e paciente. E um relacionamento de um para um com a tabela atendimento.

A tabela Profissional por sua vez tem um relacionamento de um para muitos com a tabela especialidade, bem como a tabela Paciente tem com as tabelas Departamento e Tipopaciente.

O Campo situação da tabela Agendamento define se o paciente já foi atendido ou não, desta forma o registro daquele Agendamento e do Atendimento não será mais mostrado nas telas.

4.3 APRESENTAÇÕES DO SISTEMA

Na Figura 3 é exibida a tela de *login* do sistema. O operador da aplicação deve informar usuário e senha para que assim possa prosseguir.

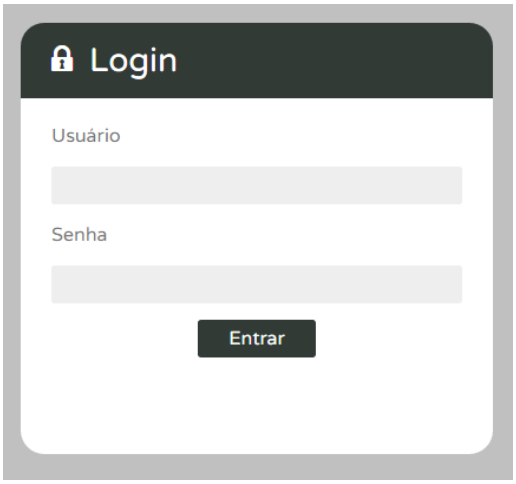

Figura 3 – Tela de Login da Aplicação

Na Figura 4 é exibida a tela inicial do sistema, pode-se observar o menu que permite acesso aos módulos de cadastro, agendamento, atendimento e os relatórios. Estes menus permitem acesso a cada uma das telas de seus respectivos módulos.

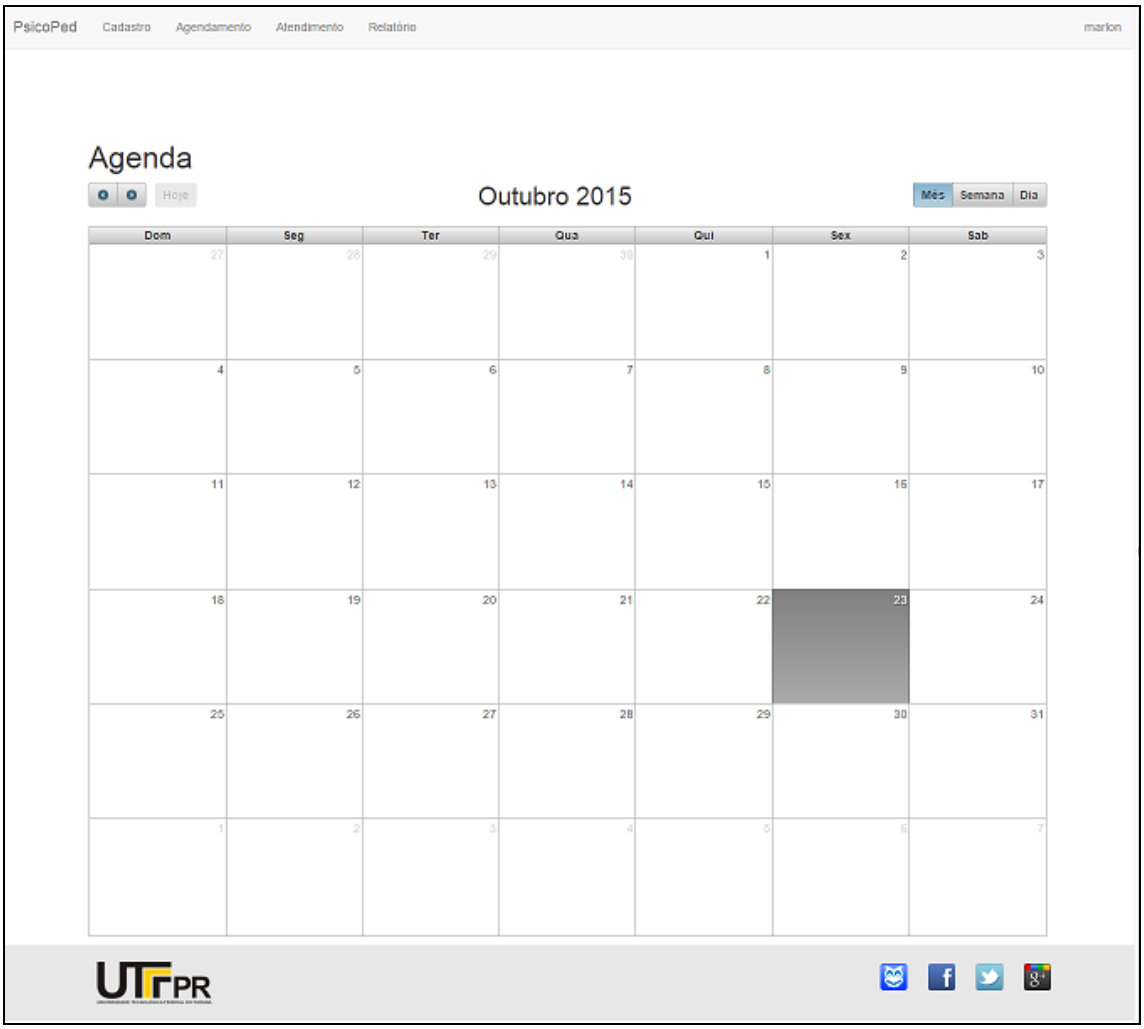

Figura 4 - Tela Inicial do Sistema

4.3.1 Módulo de Cadastros

O módulo de cadastros refere-se às principais telas de cadastro do sistema, sendo elas: departamento, especialidade, paciente, profissional, tipo de paciente e usuário. A Figura 5 apresenta os itens do menu de Cadastros.

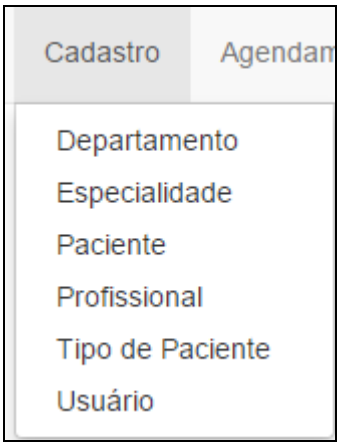

Figura 5 - Módulo de Cadastros

A tela de cadastro de Departamento/Setor tem como principal objetivo cadastrar os departamentos/setores da UTFPR, para que desta forma seja mais fácil identificar de qual setor é o paciente. Ao ser acessada esta funcionalidade trará ao usuário uma grade com as informações do Departamento/Setor e sua Sigla. Os botões Novo, Editar e Excluir são responsáveis pelo CRUD (*Create, Read, Update, Delete*) da tela. Essa tela é mostrada na Figura 6.

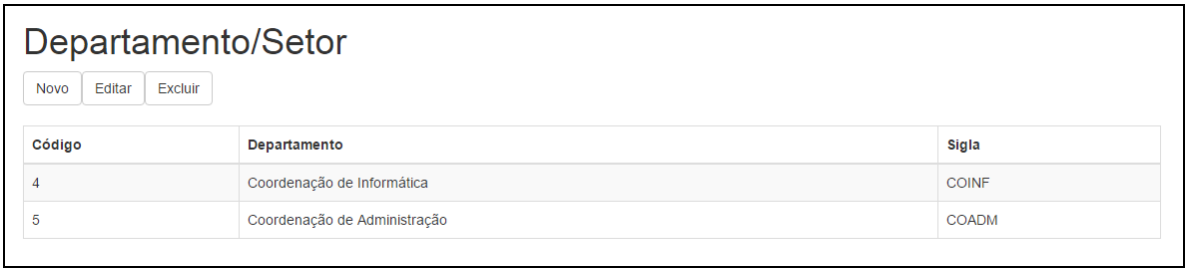

Figura 6 - Listagem de Departamento(s)/Setor(es)

Ao clicar em editar e/ou excluir se o usuário não selecionou uma linha da grade será apresentada uma mensagem de validação, conforme mostrado na Figura 7.

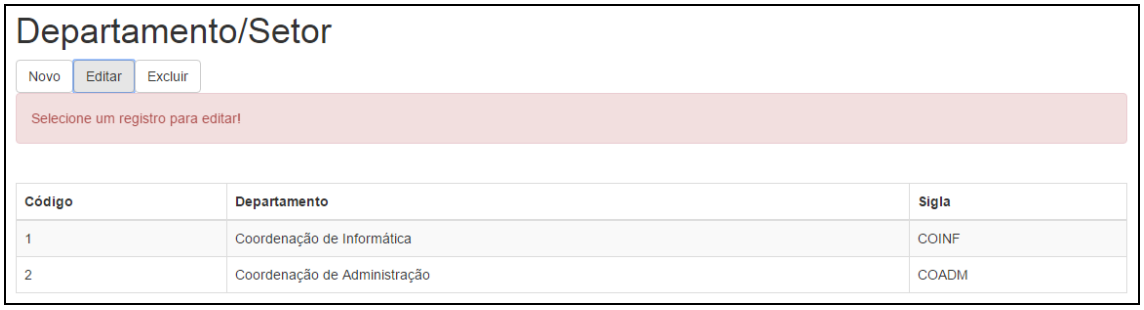

Figura 7 – Validação Departamento/Setor

Ao executar as ações do botão Novo e/ou Editar é apresentada a tela de manutenção das informações do Departamento/Setor, a qual é exibida na Figura 8. Nesta tela são apresentados os campos Descrição e Sigla, ambos sendo obrigatórios e os botões Salvar e Cancelar.

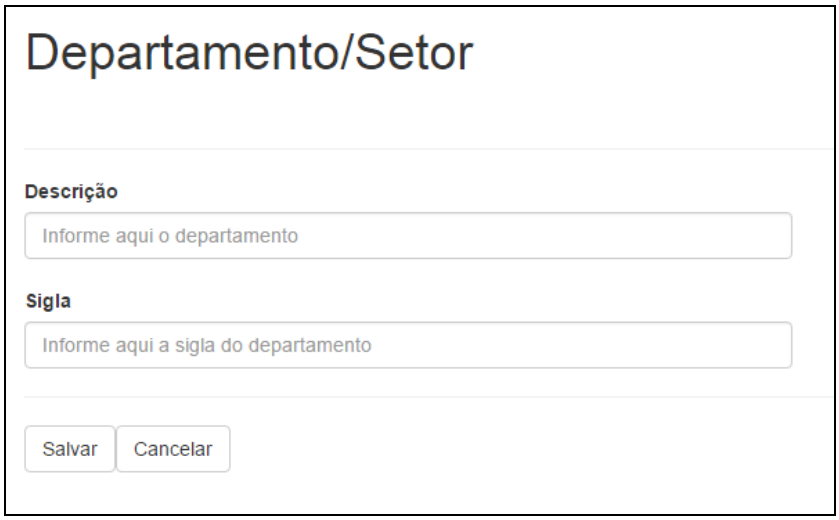

Figura 8 - Cadastro de Departamento/Setor

Clicando em Salvar as informações serão persistidas em banco e clicando em Cancelar as informações serão excluídas dos campos para que o usuário possa digitá-las novamente.

A tela de cadastro de Especialidade tem como principal objetivo cadastrar as especialidades dos profissionais. Ao ser acessada esta funcionalidade será exibida ao usuário uma grade com as informações da Especialidade, a qual pode ser visualizada na Figura 9. Os botões Novo, Editar e Excluir são responsáveis pelo CRUD da tela.

Figura 9 - Listagem de Especialidades

Seguindo o mesmo modelo da listagem de Departamento/Setor as ações de editar e/ou excluir são validadas e caso o usuário não tenha selecionado uma linha da grade será apresentada uma mensagem de validação. Ao excluir um registro será apresentada uma mensagem pedindo ao usuário se tem certeza que deseja excluir

aquele agendamento, como pode ser visto na Figura 10, caso escolha o botão OK, o mesmo será excluído.

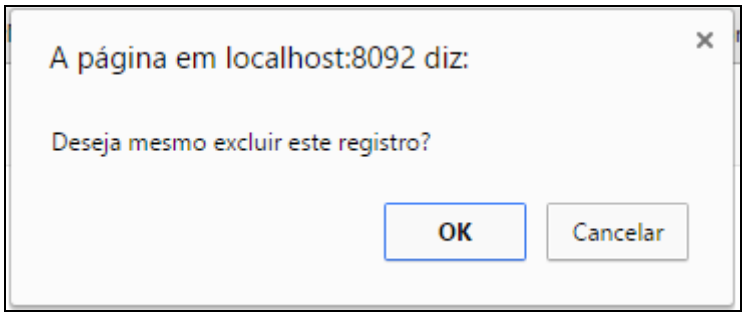

Figura 10 – Mensagem de Validação ao excluir um registro

Ao clicar no botão Novo e/ou Editar é apresentada a tela de manutenção das informações da Especialidade, conforme mostra a Figura 11. Nesta tela é apresentado o campo Descrição sendo este obrigatório e os botões Salvar e Cancelar.

Figura 11 - Cadastro de Especialidade

Clicando em Salvar as informações serão persistidas em banco e clicando em Cancelar as informações serão apagadas dos campos para que o usuário possa digitá-las novamente.

A tela de cadastro de Paciente bem como as telas anteriores tem como principal objetivo cadastrar o paciente que é quem será atendido, podendo este ser, por exemplo, um aluno, ou professor. Essa tela é exibida na Figura 12.

Figura 12 - Listagem de Pacientes

Ao clicar no botão Novo e/ou Editar é apresentada a tela de manutenção das informações do Paciente, como exibido na Figura 13. Nesta tela são apresentados os campos Nome, Departamento e Tipo de Paciente, sendo ambos obrigatórios, e os botões Salvar e Cancelar.

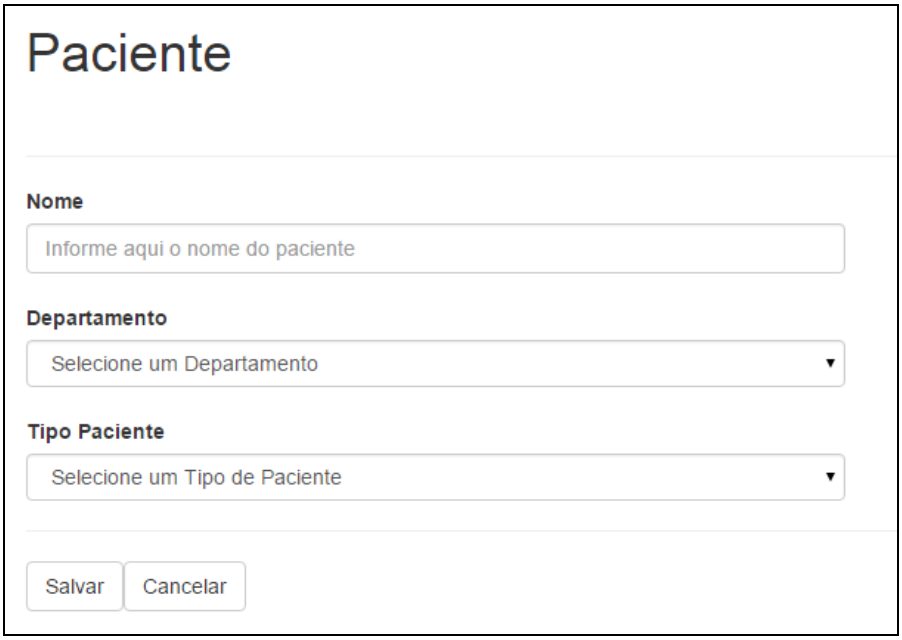

Figura 13 - Cadastro de Paciente

A tela de cadastro Usuário tem como objetivo cadastrar os usuários que irão operar o sistema e realizar novos agendamentos. Essa tela é mostrada na Figura 14.

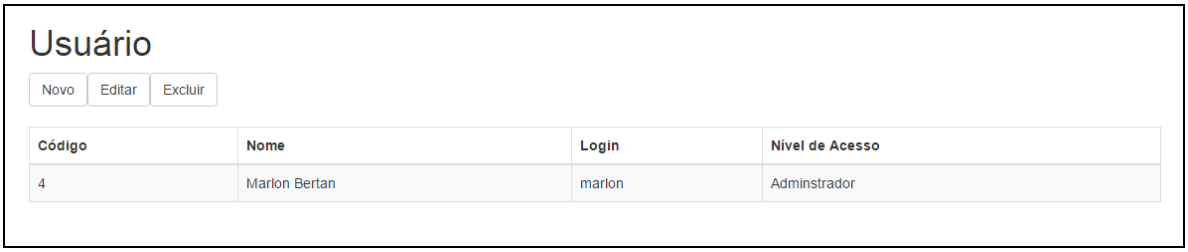

Figura 14 - Listagem de Usuários

Ao clicar no botão Novo e/ou Editar é apresentada a tela de manutenção das informações Usuário. Nesta tela são apresentados os campos Nome, Login, Senha, Confirmar Senha e Nível de Acesso, podendo neste último ser Administrador ou Usuário convencional, e os botões Salvar e Cancelar, conforme a Figura 15.

Figura 15 - Cadastro de Usuários

Clicando em Salvar as informações serão persistidas em banco e clicando em Cancelar as informações serão excluídas dos campos para que o usuário possa digitá-las novamente. É importante frisar que apenas um usuário com Nível de Acesso igual a Administrador poderá cadastrar novos usuários.

Ao editar, os campos Senha e Confirmar Senha serão apresentados em branco, para que o usuário digite uma nova senha.

4.3.1 Módulo de Agendamento

A Figura 16 mostra o menu por meio do qual é acessada a principal tela do sistema, a tela de Agendamentos.

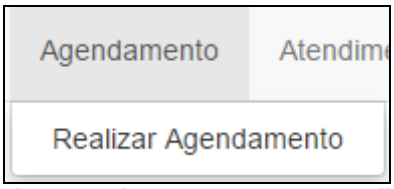

Figura 16 - Módulo Agendamento – Menu Realizar Agendamento

A tela de agendamento tem como principal objetivo agendar consultas. Essa tela une dentro de si os cadastros de usuário, paciente e profissional. Ao ser acessada esta funcionalidade trará ao usuário um calendário com as informações do Profissional a quem a consulta foi agendada, do Paciente e o usuário que realizou o agendamento, juntamente com a data da consulta e a hora que a mesma está marcada. Essa tela pode ser visualizada na Figura 17.

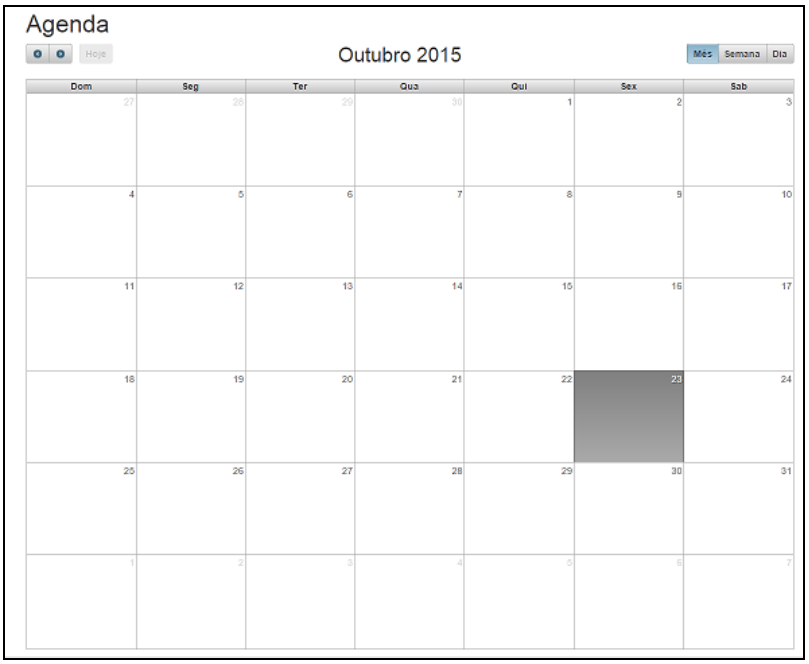

Figura 17 - Listagem de Agendamentos

Na Figura 18 é apresentada a tela de manutenção, inserção e exclusão de agendamentos, na qual os usuários devem informar os dados do Profissional, Paciente, Data e Hora da consulta. É implicitamente salvo também o usuário que realizou o agendamento. Após salvar um novo agendamento o mesmo ficará disponível na listagem de atendimentos para que desta forma o paciente seja atendido.

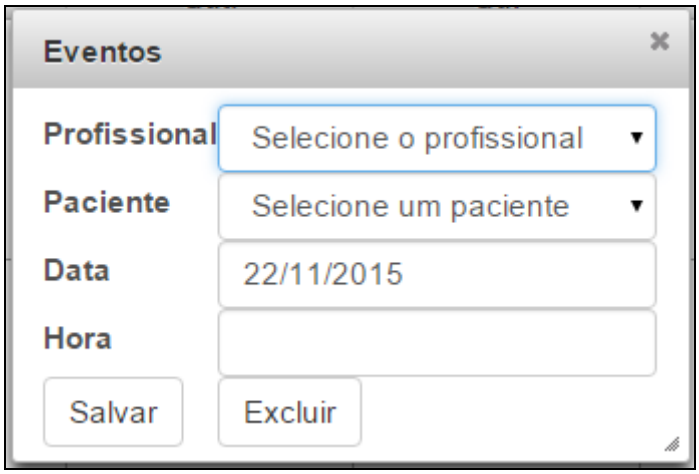

Figura 18 - Manutenção e Inserção de Agendamentos

4.3.3 Módulo de Atendimento

A Figura 19 mostra o menu por meio do qual é acessada a funcionalidade de Atendimento.

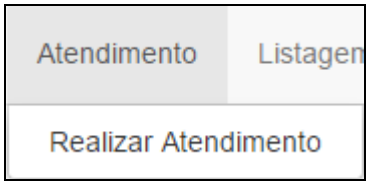

Figura 19 - Módulo Atendimento – Menu Realizar Atendimento

A tela de atendimento tem como principal objetivo atender as consultas previamente agendadas. Os atendimentos podem ser feitos apenas para consultas agendadas, ou seja, não é possível realizar um atendimento sem que o mesmo tenha sido previamente agendado.

Ao ser acessada esta funcionalidade trará ao usuário uma grade com as informações do Profissional e do Paciente, bem como a data e a hora do atendimento, como é mostrado na Figura 20.

Figura 20 - Listagem de Atendimentos

O botão Atender levará o usuário para a tela na qual será registrado o Atendimento. Nessa tela são carregadas as informações que foram cadastradas no

Agendamento, sendo elas as informações do Profissional, Paciente, Data Agendamento, Hora Agendamento e o Prontuário. No campo Prontuário as informações dos sintomas e diagnósticos serão armazenados, conforme mostra a Figura 21.

Figura 21 – Geração de um novo atendimento

4.3.4 Módulo de Listagens

A Figura 22 mostra o menu por meio do qual é acessada a funcionalidade de Listagem de Atendimentos.

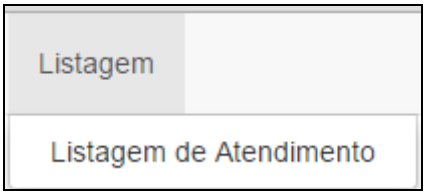

Figura 22 – Módulo Listagem – Menu Listagem de Atendimento

A tela para pesquisar atendimentos tem como objetivo mostrar o histórico dos pacientes que já foram atendidos, ou que ainda vão ser atendidos.

Ao ser acessada esta funcionalidade trás uma tela com um campo para a seleção de profissional, um campo para a seleção de paciente, um campo para a

seleção da situação que se encontra o atendimento, ou seja, aberto ou fechado, uma data inicial e uma data final, como é mostrado na Figura 23.

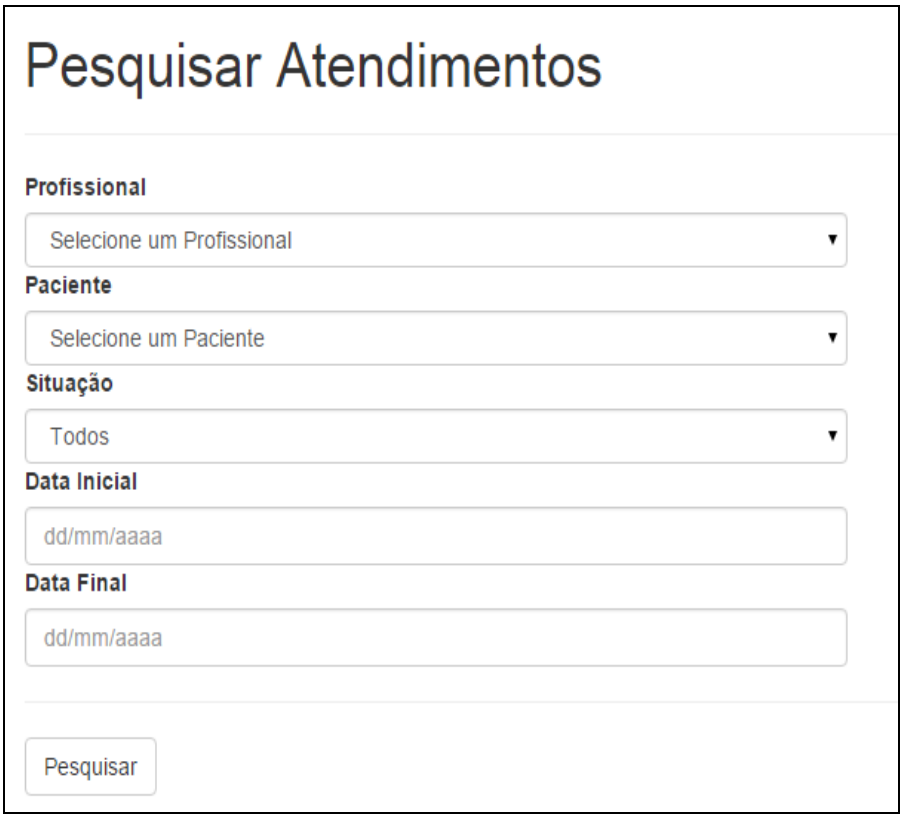

Figura 23 – Pesquisa de Atendimentos

Ao clicar no botão pesquisar a tela serão listados todos os atendimentos que foram encontrados com as informações obtidas nos campo, esta tela funcionará independente da quantidade de campos que o usuário informar, ou seja, poderá realizar pesquisas específicas, assim como poderá realizar pesquisas mais abrangentes. O click do botão pesquisar, leva para a tela de listagem de atendimentos, como pode ser visto na Figura 24.

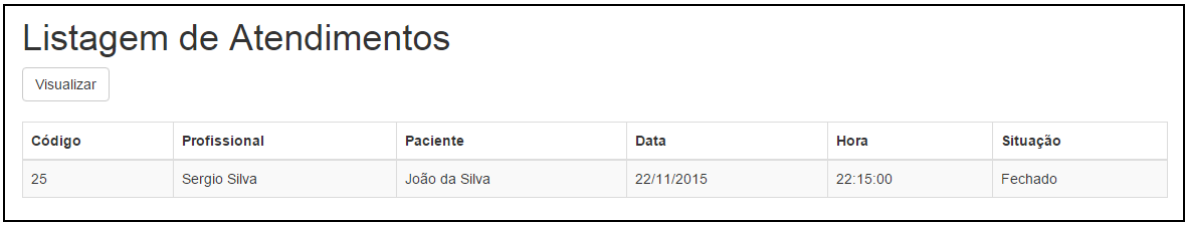

Figura 24 – Listagem de Atendimentos

A tela de listagem de atendimentos contem uma grade que trás as informações solicitadas do atendimento que foi pesquisado, após selecionar qual

atendimento deseja visualizar poderá ser clicado no botão visualizar, o qual mostrará a tela de atendimento, porém nesta tela, o usuário poderá apenas consultar o atendimento e verificar o histórico que foi salvo para o mesmo. Como pode ser visto na Figura 25.

Figura 25 – Consulta de um atendimento

A tela de consulta de atendimento mostrará todos os detalhes do atendimento que foi escolhido pelo usuário, podendo assim acompanhar o prontuário do paciente. A tela contem também um botão voltar, o qual retorna para a tela de listagem de atendimentos, que é mostrado na Figura 24.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Para o desenvolvimento do sistema foram utilizadas as tecnologias Java EE, Java Server Faces (JSF), JavaScript e Java Persistence API (JPA).

Na Listagem 1 são apresentadas as importações das classes JSF juntamente com o *template* padrão utilizado em todas as telas. No *template* padrão são criados

os menus, o rodapé e os contêiners nos quais serão inseridos os fontes das demais funcionalidades.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      xmlns:f="http://java.sun.com/jsf/core"
      template="../../template/common/layout.xhtml">

      <ui:define name="content">

      </ui:define>
</ui:composition>
```

Listagem 1 – Template Padrão

A Listagem 2 contém as *tags* Html, Head e Body, que não são apresentadas explicitamente nas demais páginas, fornecendo reuso de código fonte. Desta forma o “header.xhtml” e o “content.xhtml” não precisam ser reescritos em todas as páginas.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">

      <h:head>
            <h:outputStylesheet name="common_style.css" library="css" />
            <h:outputStylesheet name="bootstrap.min.css" library="css" />
            <h:outputStylesheet name="navbar.css" library="css" />
            <h:outputScript library="js" name="jquery.min.js" />
            <h:outputScript library="js" name="bootstrap.min.js" />
            <h:outputScript library="js" name="ie10-viewport-bug-workaround.js" />
            <h:outputScript library="js" name="jquery.maskedinput.js" />
            <title>Sistema de Agendamento Psico-Pedagógico</title>
      </h:head>
      <h:body>
            <div id="page">
                  <div id="header">
                        <ui:insert name="header">
                              <ui:include src="header.xhtml" />
                        </ui:insert>
                  </div>
                  <h:form prependId="false">
                        <div id="content">
                              <ui:insert name="content">
                                    <ui:include src="content.xhtml" />
                              </ui:insert>
                        </div>
                  </h:form>
            </div>
      </h:body>
</html>
```

Listagem 2 – Layout Genérico das Páginas

A Listagem 3 mostra a criação dos botões que são mostrados na página de listagem de Pacientes, juntamente com suas estilizações.

```
<div class="container">
        <div class="starter-template">
              <h1>Paciente</h1>
              <h:outputLink styleClass="btn btn-default" value="cadastro.xhtml">
                    <h:outputText value="Novo" />
              </h:outputLink>
              <h:commandButton styleClass="btn btn-default" value="Editar"
                    id="editar" action="#{pesquisaPacienteBean.editar()}" />
              <h:commandButton styleClass="btn btn-default" value="Excluir"
                    id="excluir" action="#{pesquisaPacienteBean.excluir()}" />
              <div id="divErro" class="alert alert-danger" style="display: none;"/>
       ...
       </div>
</div>
```

Listagem 3 - Criação dos botões Novo, Editar e Excluir

Na Listagem 4 é criada a tabela na qual são apresentados os dados do Paciente, bem como suas colunas, que por sua vez são ligadas aos ManagedBeans. Por último a declaração do JavaScript que foi utilizado.

```
...
<div class="spacer">
      <br/>
</div>
<h:inputHidden value="#{pesquisaPacienteBean.codigo}" id="pk" />
      <div id="divErro" />
      <h:dataTable id="tabela" value="#{pesquisaPacienteBean.lista}"
            var="item"
            styleClass="table table-striped table-hover table-bordered display">
            <h:column>
                  <f:facet name="header">
                        <h:outputText value="Código" />
                  </f:facet>
                  <h:outputText value="#{item.codigo}" />
            </h:column>
            <h:column>
                  <f:facet name="header">
                        <h:outputText value="Nome" />
                  </f:facet>
                  <h:outputText value="#{item.nome}" />
            </h:column>
            <h:column>
                  <f:facet name="header">
                        <h:outputText value="Departamento" />
                  </f:facet>
                  <h:outputText value="#{item.departamento.descricao}" />
            </h:column>
            <h:column>
                  <f:facet name="header">
                        <h:outputText value="Tipo Paciente" />
                  </f:facet>
                  <h:outputText value="#{item.tipopaciente.descricao}" />
            </h:column>
      </h:dataTable>
      <div class="spacer" />
      <h:outputScript library="js" name="selecaoCodigo.js" />
...
```

Listagem 4 - Criação da Listagem de Pacientes

Na Listagem 5 é apresentado o código fonte do JavaScript responsável pela seleção das linhas das grades de suas respectivas listagens e validações, disparadas ao clicar nos botões editar, excluir e atender.

```
var id = "";
$('table tbody tr').click(function(event){
	id = event.currentTarget.cells[0].innerText;

	var i=$("tr[style]");
	if (i.length > 0) {
		var c = i[0];
		c.removeAttribute("style", "");
	}
	event.currentTarget.setAttribute("style", "background-color: #000; color: #FFF;");
	document.getElementById('pk').value = id;
});

$('#editar').click(function(event){
	if (id == 0){
		exibeResultado("Selecione um registro para editar!");
		event.stopPropagation();
		return false;
	}
});

$('#excluir').click(function(event){
	if (id == 0){
		exibeResultado("Selecione um registro para remover!");
		event.stopPropagation();
		return false;
	}else {
		var r = confirm("Deseja mesmo excluir este registro?");
		if (r == false){
			event.stopPropagation();
			return false;
		}
	}
});

$('#atender').click(function(event){
	if (id == 0){
		exibeResultado("Selecione um registro para atender!");
		event.stopPropagation();
		return false;
	}
});

function exibeResultado(mensagem){
     $('#divErro').html(mensagem);
     $('#divErro').fadeIn('fast');
     $('#divErro').fadeOut(2500);
}
```

Listagem 5 - JavaScript responsável pela seleção de uma linha da tabela e validações

O código fonte exibido na Listagem 6 refere-se à criação do campo de texto “nome” do cadastro de pacientes, sendo este ligado ao ManagedBean específico. O ManagedBean é a classe responsável por “delegar” funções específicas para cada formulário da aplicação.

```
...
<div class="row">
      <div class="col-md-6">
              <h:outputLabel for="nome" value="Nome" class="control-label" />
              <h:inputText value="#{listaPacienteBean.paciente.nome}" id="nome"
                           class="form-control"
                           pt:placeholder="Informe aqui o nome do paciente"
                           validatorMessage="Nome inválido."
                           requiredMessage="É necessário informar um Nome.">
              </h:inputText>
              <h:message id="m_nome" for="nome" />
      </div>
</div>
...
```

Listagem 6 - Campo Nome, do cadastro de Pacientes

A Listagem 7 refere-se à criação do *combo* de departamentos que é utilizado na tela de manutenção do Paciente. Este campo busca suas informações do ManagedBean listaPacienteBean, mais especificamente do objeto departamento e por meio do conversor departamentoConverter as informações são carregadas corretamente em tela.

```
<div class="row">
     <div class="col-md-6">
             <h:outputLabel for="departamento" value="Departamento"
                            class="control-label" />
             <h:selectOneMenu id="departamento" class="form-control"
                            value="#{listaPacienteBean.paciente.departamento}"
                            converter="departamentoConverter">
                        <f:selectItem itemLabel="Selecione um Departamento" itemValue="" />
                    <f:selectItems value="#{listaPacienteBean.listaDepartamentos}"
                                   var="departamento" itemValue="#{departamento}"
                                   itemLabel="#{departamento.descricao}">
                    </f:selectItems>
             </h:selectOneMenu>
     </div>
</div>
```

Listagem 7 - Campo Departamento, do cadastro de Pacientes

O departamentoConverter é responsável por realizar a conversão entre objetos para que o JSF (Java Server Faces) saiba que a informação selecionada é de um tipo específico. Para isso é criada uma classe Java que implementa a interface Converter, que por sua vez tem dois métodos abstratos. Essa classe é anotada com “@FacesConverter” para que o servidor, ao iniciar a aplicação, identifique a classe, deixando-a disponível. O código fonte do conversor é visto na Listagem 8.

```java
@FacesConverter("departamentoConverter")
public class DepartamentoConverter implements Converter {

	@Inject
	private DepartamentoDAO dao;

	public DepartamentoDAO getDao() {
		return dao;
	}

	public void setDao(DepartamentoDAO dao) {
		this.dao = dao;
	}

	@Override
	public Object getAsObject(FacesContext context, UIComponent component, String value) {
		if (value == null || value.equals("")) {
			return null;
		}
		Departamento dpto = new Departamento();
		dpto = getDao().pesquisarPorCodigo(Integer.parseInt(value));
		return dpto;
	}

	@Override
	public String getAsString(FacesContext context, UIComponent component, Object object) {
		if (object == null || object.equals("")) {
			return null;
		}
		return ((Departamento) object).getCodigo().toString();
	}
}
```

Listagem 8 – Classe Conversora de Departamentos

Na Listagem 9 é mostrada a criação dos botões Salvar e Cancelar, responsáveis diretamente pela persistência das informações no banco de dados.

```xml
<div class="form-group">
	<div class= "row">
		<div class="col-md-12">
			<h:commandButton action="#{listaPacienteBean.salvar()}"
					value="Salvar" class="btn btn-default" />
			<h:commandButton action="#{listaPacienteBean.cancelar()}"
					value="Cancelar" class="btn btn-default" />
		</div>
	</div>
</div>
```

Listagem 9 - Criação dos botões Salvar e Cancelar do Cadastro de Pacientes

A Listagem 10 mostra o código do botão cancelar, que realiza a ação de chamar o método *setter* do objeto Paciente criando uma nova instância do mesmo. Em seguida a página é redirecionada para o cadastro de pacientes.

```java
@Override
public void cancelar() throws IOException {
	setPaciente(new Paciente());
	redirect("/utfpr/paciente/cadastro.xhtml");
}
```

Listagem 10 - Método Cancelar do Bean do Paciente

A Listagem 11 mostra a ação realizada pelo método Salvar. Ao ser invocado, este método executa o comando alterar do controlador do Paciente, passando por parâmetro um objeto do tipo Paciente para que o mesmo seja persistido no banco de dados. Após isso, a página é redirecionada para a listagem de pacientes.

```
@Override
public void salvar() {
      controller.alterar(paciente);
      redirect("/utfpr/paciente/lista.xhtml");
}
```

Listagem 11 - Método Salvar do Bean do Paciente

O método editar, como mostra a Listagem 12, apenas redireciona o usuário para a tela de manutenção.

```
@Override
public void editar() {
       redirect("/utfpr/paciente/cadastro.xhtml");
}
```

Listagem 12 - Método Editar do Bean de Paciente

O método excluir como mostrado na Listagem 13, realizará uma pesquisa pelo código do paciente. Então será executado o método excluir do controlador do paciente, passando por parâmetro o objeto paciente que acabou de ser retornado pela pesquisa. Por fim, o usuário é redirecionado para a página de listagem de pacientes.

```
@Override
public void excluir() {
       paciente = controller.pesquisarPorCodigo(getCodigo());
       controller.excluir(paciente);
       redirect("/utfpr/paciente/lista.xhtml");
}
```

Listagem 13 – Método Excluir do Bean de Paciente

Na Listagem 14 são apresentados os controladores que são utilizados no "*ManagedBean"* do Paciente. Todos estão anotados com "@Inject" que faz com que o *Contexts and Dependency Injection* (CDI) crie uma nova instância para cada um, deixando-os assim pronto para o uso. O CDI é a especificação Java EE responsável por realizar a injeção de dependências entre as classes da aplicação.

```
@Inject
private PacienteController controller;

@Inject
private TipopacienteController tipopacienteController;

@Inject
private DepartamentoController departamentoController;
```

Listagem 14 – Utilizando Injeção de Dependência com CDI

A Listagem 15 exibe o método “inicializar()”, que tem como função instanciar o objeto paciente e buscar informações para as listagens de Tipo de Paciente e Departamento, realizando também uma verificação para então realizar a pesquisa para carregar as informações do paciente. O método “getCodigo” é invocado de sua classe ancestral.

O método “inicializar()” é anotado com “@PostConstruct” que garante que este método seja chamado junto a inicialização dos serviços do contêiner para o *bean*.

```
@PostConstruct
private void inicializar(){
	setPaciente(new Paciente());
	setListaTipopacientes(getTipopacienteController().listar());
	setListaDepartamentos(getDepartamentoController().listar());
	if (getCodigo() != null) {
		paciente = controller.pesquisarPorCodigo(getCodigo());
	}
}
```

Listagem 15 – Método Inicializar do Bean do Paciente

A Listagem 16 apresenta a classe paciente. Utilizando as anotações JPA @Entity, para informar que a mesma é uma entidade e @Table para definir a qual tabela ela se refere.

```
@Entity
@Table(name = "paciente")
public class Paciente implements Serializable{
	private static final long serialVersionUID = 1L;
```

Listagem 16 – Entidade do Paciente – Anotações de Tabela

A Listagem 17 mostra o campo código contendo a anotação de “@Id” para informar que é a chave primária da tabela. A anotação “@SequenceGenerator” para definir que sejam geradas as sequências da chave primária automaticamente,

“@GeneratedValue” que se refere ao campo código como sendo chave da tabela. Obrigando, dessa forma, a criação de um construtor sem argumentos.

```
@Id
@SequenceGenerator(name = "paciente_id_paciente_seq",
                   sequenceName="paciente_id_paciente_seq",
                   allocationSize = 1)
@GeneratedValue(generator = "paciente_id_paciente_seq",
                  strategy = GenerationType.IDENTITY)
@Column(name = "id_paciente")
private Integer codigo;
```

Listagem 17 – Entidade do Paciente – Anotações de identificador da tabela

A Listagem 18 apresenta a junção entre as tabelas “Paciente” e “Tipopaciente”, através da cláusula “@JoinColumn” no campo “id_tipopaciente”. A anotação “@ManyToOne” se refere ao tipo do relacionamento entre as tabelas, com um “FetchType” igual a “EAGER” para que a propriedade seja sempre carregada.

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "id_tipopaciente")
private Tipopaciente tipopaciente;
```

Listagem 18 – Entidade do Paciente – Junção entre Tabelas

A Listagem 19 mostra a interface “GenericDAO” que recebe um objeto “T”. Utilizando o conceito de Generics esse objeto pode ser de qualquer tipo, evitando o uso de typeCast, que é a conversão de um tipo de dados para outro, propiciando, também, maior reaproveitamento de código. Essa interface contém os métodos salvar, alterar, deletar, pesquisarPorCodigo e listar. Esses métodos são utilizados nas classes que implementam esta interface.

```
public interface GenericDAO<T> {

     T salvar(T t);

     void alterar(T t);

     void deletar(Integer id);

     T pesquisarPorCodigo(Integer id);

     List<T> listar();
}
```

Listagem 19 – Interface GenericDAO

A Listagem 20 mostra a utilização da classe “GenericDAO”, esta por sua vez é implementada pela classe “GenericDAOImp” e assim utilizando seus métodos.

Essa classe contém um EntityManager, que é o responsável por persistir as informações no banco de dados e gerenciar o ciclo de vida das entidades anotada

com “@PersistenceContext”, que é uma espécie de contêiner que guarda as entidades para que o EntityManager não precise buscar toda vez o mesmo objeto no banco de dados. O método construtor é anotado com “@SuppressWarnings” que serve basicamente para silenciar as advertências que ocorrem quando se manipula um objeto sem especificar seu tipo, que é exatamente o que ocorre neste método.

Temos também o método deletar que recebe um número inteiro por parâmetro e o utiliza para excluir o objeto (classe) (que foi referenciado à variável “clazz” no método construtor) do banco de dados.

O método listar, a partir do criteriaBuilder, que é uma maneira de realizar consultas no banco de dados, retorna uma lista de objetos do tipo genérico. O método salvar recebe um objeto por parâmetro e devolve um objeto do mesmo tipo. Para persistir a informação no banco de dados ele utiliza do método “persist” do “EntityManager”. O método alterar recebe um objeto por parâmetro e por ,meio do “EntityManager” aciona o método “merge”, que verifica se a informação deve ser inserida ou editada. O método “pesquisarPorCodigo” recebe um número inteiro por parâmetro e o utiliza para retornar um objeto também por meio do “EntityManager”. A classe apresenta também os métodos *“getter”* e *“setter”* das variáveis declaradas.

```java
public abstract class GenericDaoImpl<T> implements GenericDAO<T> {

	@PersistenceContext
	protected EntityManager em;
	private Class<T> clazz;

	@SuppressWarnings({ "unchecked", "rawtypes" })
	public GenericDaoImpl() {
		Type t = getClass().getGenericSuperclass();
		ParameterizedType pt = (ParameterizedType) t;
		clazz = ((Class) pt.getActualTypeArguments()[0]);
	}

	protected EntityManager getEntityManager() {
		return em;
	}

	protected void setEntityManager(EntityManager entityManager) {
		this.em = entityManager;
	}

	public void deletar(final Integer id) {
		try {
			getEntityManager().remove(em.getReference(clazz, id));
		} catch (Exception e) {
			System.out.print("Erro ao excluir!");
		}
	}

	public List<T> listar() {
		CriteriaBuilder builder = em.getCriteriaBuilder();
		CriteriaQuery<T> cQuery = builder.createQuery(getTypeClass());
		Root<T> from = cQuery.from(getTypeClass());
		CriteriaQuery<T> select = cQuery.select(from);
		List<Predicate> predicates = new ArrayList<>();
		select.where(predicates.toArray(new Predicate[] {}));
		select.orderBy(builder.asc(from.get("codigo")));
		TypedQuery<T> listQuery = em.createQuery(select);
		return listQuery.getResultList();
	}

	private Class<T> getTypeClass() {
		return clazz;
	}

	@TransactionAttribute
	public T salvar(final T t) {
		try {
			getEntityManager().persist(t);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return t;
	}

	public void alterar(final T t) {
		getEntityManager().merge(t);
	}

	public T pesquisarPorCodigo(Integer id){
		return (T) getEntityManager().find(clazz, id);
	}
}
```

Listagem 20 – Classe GenericDAOImpl

Na Listagem 21 é apresentada a interface “PacienteDAO”. Essa interface é herdada da classe “GenericDAO” que foi mostrada na listagem 19, sendo ela do tipo Paciente.

```java
public interface PacienteDAO extends GenericDAO<Paciente>{
}
```

Listagem 21 – Interface PacienteDAO

A classe “PacienteDAOImp”, mostrada na Listagem 22, herda suas funcionalidades da classe “GenericDAOImpl” passando o objeto Paciente como tipo da classe e também implementa a interface “PacienteDAO”, apresentada na Listagem 21.

```
@Stateless(name = "pacienteDAO")
public class PacienteDAOImpl extends GenericDaoImpl<Paciente> implements PacienteDAO{
}
```

Listagem 22 – Classe PacienteDAOImpl

Na Listagem 23 é apresentada a interface “GenericController”. Essa interface é genérica e contém os métodos excluir, salvar e alterar, que recebem por parâmetro um objeto genérico. O método listar que retorna uma lista genérica e o método “pesquisarPorCodigo” que recebe um número inteiro por parâmetro e retorna um objeto genérico.

```
public interface GenericController<T> {

    void excluir(T t);
    void salvar(T t);
    void alterar(T t);
    List<T> listar();
    T pesquisarPorCodigo(Integer id);
}
```

Listagem 23 – Interface GenericController

Na Listagem 24 é mostrada a interface “PacienteController” que é herdada da classe “GenericController”, que foi apresentada na Listagem 23, sendo ela do tipo paciente.

```
public interface PacienteController extends GenericController<Paciente>{
}
```

Listagem 24 – Interface PacienteController

Na Listagem 25 é apresentada a classe “PacienteControllerImpl”. Essa classe implementa a interface “PacienteController”, sobrescrevendo os métodos excluir, salvar, alterar, listar e pesquisarPorCodigo. Nessa classe é declarada a variável “dao”, do tipo “PacienteDAO” que é utilizada em todos os métodos desta classe para referenciar os métodos da interface “GenericDAO”, que será implementada pela interface “PacienteDAO”, que por sua vez será implementada pela classe “PacienteDAOImpl”, que herda a classe “GenericDAOImpl” que terá os métodos necessários para que as informações sejam persistidas no banco de dados.

```java
@Stateless(name = "pacienteController")
public class PacienteControllerImpl implements PacienteController{

	@EJB
	private PacienteDAO dao;

	@Override
	public void excluir(Paciente t) {
		dao.deletar(t.getCodigo());
	}

	@Override
	public void salvar(Paciente t) {
		dao.salvar(t);
	}

	@Override
	public void alterar(Paciente t) {
		dao.alterar(t);
	}

	@Override
	public List<Paciente> listar() {
		return dao.listar();
	}

	@Override
	public Paciente pesquisarPorCodigo(Integer id) {
		return dao.pesquisarPorCodigo(id);
	}
}
```

Listagem 25 - Classe PacienteControllerImp

As Listagens 26, 27 e 28 apresentam o arquivo “pom.xml”. Esse arquivo é utilizado pelo *framework* Maven, para gerenciar as dependências dos projetos java, tornando muito mais fácil e prático o desenvolvimento. Basta apenas informar a dependência do *framework* que está sendo usado no arquivo “pom.xml” que o Maven se encarregará de fazer o download das dependências necessárias.

Na Listagem 26 é ilustrado o cabeçalho do arquivo “pom.xml”, contendo as informações do projeto que foi criado utilizando o Maven e suas propriedades.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>br.edu.utfpr</groupId>
	<artifactId>posjava</artifactId>
	<version>1.0</version>
	<packaging>war</packaging>

	<name>posjava</name>

	<properties>
		<endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>
```

Listagem 26 - Pom.xml, informações do projeto

Na Listagem 27 são apresentadas todas as dependências utilizadas para o desenvolvimento da aplicação, como o Java Server Faces e o Hibernate, que é um *framework* utilizado para trabalhar com JPA.

```
<dependencies>
        <dependency>
                <groupId>com.sun.faces</groupId>
                <artifactId>jsf-api</artifactId>
                <version>2.1.0-b03</version>
        </dependency>
        <dependency>
                <groupId>com.sun.faces</groupId>
                <artifactId>jsf-impl</artifactId>
                <version>2.1.0-b03</version>
        </dependency>
        <dependency>
                <groupId>javax</groupId>
                <artifactId>javaee-web-api</artifactId>
                <version>7.0</version>
                <scope>provided</scope>
        </dependency>
        <dependency>
                <groupId>javax</groupId>
                <artifactId>javaee-web-api</artifactId>
                <version>7.0</version>
                <scope>provided</scope>
        </dependency>
        <dependency>
                <groupId>org.json</groupId>
                <artifactId>json</artifactId>
                <version>20140107</version>
        </dependency>
        <dependency>
                <groupId>javax.inject</groupId>
                <artifactId>javax.inject</artifactId>
                <version>1</version>
                <scope>provided</scope>
        </dependency>
        <dependency>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-log4j12</artifactId>
                <version>1.7.5</version>
        </dependency>
        <dependency>
                <groupId>org.javassist</groupId>
                <artifactId>javassist</artifactId>
                <version>3.18.1-GA</version>
        </dependency>
        <dependency>
                <groupId>jstl</groupId>
                <artifactId>jstl</artifactId>
                <version>1.2</version>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-entitymanager</artifactId>
                <version>4.3.5.Final</version>
                <scope>provided</scope>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-validator</artifactId>
                <version>5.1.1.Final</version>
        </dependency>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.11</version>
                <scope>test</scope>
        </dependency>
        <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>5.2</version>
        </dependency>
</dependencies>
```

Listagem 27 – Pom.xml, dependências do projeto

Na Listagem 28 estão listados os *plugins* do próprio Maven que são responsáveis pelo *download* das dependências do projeto.

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>2.6</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Listagem 28 – Plugins do Maven

5 CONCLUSÃO

O objetivo deste trabalho foi implementar um aplicativo *web* para realizar agendamentos e controlar consultas de dois setores específicos da UTFPR, Câmpus Pato Branco, o NAPNE/NUAPE. O aplicativo foi desenvolvido utilizando diversas ferramentas e tecnologias, as quais contribuíram de forma a agilizar o processo de desenvolvimento.

Embora seja muito trabalhoso e por vezes até um pouco complicado trabalhar com uma interface *web* ao final é recompensador. As tecnologias usadas para o desenvolvimento do aplicativo atenderam as expectativas. As principais tecnologias utilizadas foram o JavaScript e Bootstrap, que foram empregadas para o desenvolvimento do lado cliente da aplicação. E as tecnologias JSF, JPA, CDI foram utilizadas para o desenvolvimento do lado servidor do projeto.

De maneira geral, pode-se verificar com este trabalho como todas as disciplinas foram importantes na construção de uma base sólida para o bom desenvolvimento do projeto.

Este trabalho deve contribuir para a mobilidade e maior facilidade no Agendamento/Atendimento dos setores NAPNE/NUAPE da UTFPR, tornando mais fácil e rápido o dia-a-dia no ambiente de trabalho desses profissionais.

Como trabalhos futuros existem vários recursos que podem ser aprimorados e também melhorados no decorrer do uso do projeto. Entre esses recursos está envio de *e-mail* para a confirmação da consulta. Também podem ser complementados os cadastros com outras informações que sejam relevantes, a criação de novos cadastros e acréscimo de funcionalidades e de relatórios para um *feedback* mais completo da situação de cada paciente e de cada atendimento.

REFERÊNCIAS

BOZZON, Alessandro; COMAI, Sara; FRATERNALI, Pietro; CARUGHI, Giovanni Toffetti. **Conceptual modeling and code generation for rich internet applications**. In: ICWE, ACM Press, 2006, p. 353–360.

CORDEIRO, Gilliard. **Aplicações Java para a web com JSF e JPA**. São Paulo: Casa do Código, 2012.

PANG, Zhen; WEN, Fuan; PAN, Xiwei; LUI, Cen. **Migration model for Rich Internet Applications based on PureMVC Framework**. In: International Coriference On Computer Design And Appliations (ICCDA 2010), v. 5, p.340-343.

LINAJE, Marino; PRECIADO, Juan Carlos; SÁNCHEZ-FIGUEROA, Fernando, COMAI, Sara. **Necessity of methodologies to model Rich Internet Applications**. In: 7th IEEE Int. Symposium on Web Site Evolution (WSE 2005), 2005, p. 7–13.

PRESSMAN, Roger. **Engenharia de software**. 6 ed. Rio de Janeiro: McGraw-Hill, 2006.

STEARN, Brent. XULRunner: **A new approach for developing Rich Internet Applications**. IEEE Internet Computing, v. 11, n.3, 2007, p.67-73.