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

MARCELO JOSE FORMENTINI

SISTEMA PARA GERENCIAMENTO DE ATENDIMENTOS DE SETOR MÉDICO-ODONTOLÓGICO

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

MARCELO JOSE FORMENTINI

SISTEMA PARA GERENCIAMENTO DE ATENDIMENTOS DE SETOR MÉDICO-ODONTOLÓGICO

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

Orientador: Vinícius Pegorini

PATO BRANCO
2015

SISTEMA PARA GERENCIAMENTO DE ATENDIMENTOS DE SETOR MÉDICO-ODONTOLÓGICO

Por

Marcelo José Formentini

Esta monografia foi apresentada às 15h30 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. Luisiana Yoshie Araki
Banca
UTFPR – Campus Pato Branco

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

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

RESUMO

FORMENTINI, Marcelo Jose. Sistema para gerenciamento de atendimento de setor médico-odontológico. 2015. 41 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Os registros dos atendimentos nos setores médico e odontológico da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, estão sendo realizados até então em planilhas manuais, o que dificulta o acompanhamento das informações, bem como a geração de relatórios. Devido a isso, uma aplicação foi desenvolvida visando facilitar e auxiliar no registro dos atendimentos desses setores e dos encaminhamentos realizados pelo setor médico. Um controle simples de estoque para os medicamentos utilizados também foi implementado. O sistema de gerenciamento de atendimentos dos setores médico e odontológico é uma aplicação web caracterizada como interface rica, foi desenvolvido com a utilização de tecnologia Java Enterprise Edition, JavaServer Pages, JavaServer Faces, entre outras.

Palavras-chave: Java para web. Aplicações Internet Ricas. Gerenciamento de setores médico-odontológicos.

ABSTRACT

FORMENTINI, Marcelo Jose. System for management of medical and dental care. 2015. 41 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

The records of attendance in the medical and dental sectors of the Federal Technological University of Paraná, Campus Pato Branco, are being carried out so far on manual spreadsheets, which complicates the monitoring of information as well as reporting. Because of this, an application was developed to facilitate and assist in recording the calls of these sectors and referrals made by the medical sector. A simple inventory control for those medicinal products has also been implemented. The care management system of medical and dental sectors is a web application characterized as rich interface was developed using Java Enterprise Edition technology, JavaServer Pages, JavaServer Faces, among others.

Keywords: Java web. Rich Internet Application. Management of medical and dental sectors.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

CSS	*Cascading Style Sheets*
EE	Enterprise Edition
EJB	Enterprise JavaBeans
HTML	*HiperText Markup Languagem*
JPA	*Java Persistence API*
JSF	JavaServer Faces
JSP	*JavaServer Pages*
ORM	*Object-Relational Mapping*
RIA	*Rich Internet Applications*
SAMU	Serviço de Atendimento Móvel de Urgência
SGDB	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
XML	*eXtensible Markup Language*
XUL	*XML User Interface Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho e é composto pelas considerações inicias com o contexto de uso da aplicação e sua inserção no referencial teórico que sustenta a proposta. Também são apresentados os objetivos e a justificativa do trabalho. 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 um setor de atendimento médico-odontológico com os profissionais médico, dentista e enfermeira que prestam serviços a funcionários (professores e técnicos administrativos e demais funcionários) e alunos. Esses atendimentos, quando da área médica, são imediatos à ocorrência e dependendo do tipo de ocorrência é realizado o devido encaminhamento, como, por exemplo, deslocamento ao pronto atendimento municipal ou solicitação de intervenção do Serviço de Atendimento Móvel de Urgência (SAMU). Os atendimentos de odontologia quando não urgentes podem ser agendados.

Atualmente o controle dos atendimentos realizados nesse setor éfeito através de anotações em planilhas. Isso dificulta a geração de relatórios e estatísticas para acompanhamento das atividades do setor. Verificou-se, assim, a necessidade de desenvolver um sistema que pudesse auxiliar na realização desse controle.

Visando facilitar a utilização e mesmo o acesso, optou-se pelo desenvolvimento de uma aplicação web caracterizada como de interface rica. A tela principal do aplicativo provê o sistema de registro dos atendimentos. E, assim, no atendimento da ocorrência já é possível fazer o registro da mesma e o paciente encaminhado imediatamente para o procedimento adequado ao ocorrido.

1.2 OBJETIVOS

A seguir são apresentados os objetivos deste trabalho, organizados em objetivo geral e objetivos específicos.

1.2.1 Objetivo Geral

- Implementar um sistema web para o registro dos atendimentos realizados no setor médico-odontológico da UTFPR, Câmpus Pato Branco.

1.2.2 Objetivos Específicos

- Fornecer uma maneira de facilitar o registro dos atendimentos realizados no setor.
- Agilizar a geração de relatórios e estatísticas de atendimentos.
- Facilitar o gerenciamento dos produtos em estoque e do seu consumo e da validade dos mesmos.

1.3 JUSTIFICATIVA

O desenvolvimento de um sistema web visa facilitar o acesso aos seus usuários, sendo profissionais distintos e em ambientes distintos que utilizarão o sistema. Não haverá necessidade de um controle de agenda, embora seja atendimento de serviços médicos e odontológicos, porque, em sua maioria, são atendimentos de emergência. Esses atendimentos acontecem imediatamente às suas ocorrências quando médicos e os odontológicos, embora não necessariamente de emergência, não foi considerado necessário haver controle de agenda pelos profissionais do setor.

A principal funcionalidade do sistema é o registro dos atendimentos, no sentido de computar a quantidade de atendimentos realizados por procedimento ou sintoma apresentado. Assim, a tela principal do aplicativo apresenta o registro dos

procedimentos e sintomas e o profissional escolhe o sintoma e marca “mais um” atendimento.

Esse registro facilita na geração de relatórios e estatísticas de atendimentos, não sendo mais necessário realizar contagens e elaborar planilhas manualmente.

1.4 ESTRUTURA DO TRABALHO

Este texto está estruturado em capítulos. O capítulo 2 apresenta o referencial teórico do trabalho que está centrado em aplicações Internet denominadas como ricas. O Capítulo 3 apresenta os materiais e o método utilizados no desenvolvimento do trabalho. O resultado (a modelagem e a implementação do sistema) é apresentado no Capítulo 4. As considerações finais são apresentadas no Capítulo 5, seguidas das referências bibliográficas utilizadas na elaboração do texto.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho, que é centrado em aplicações Internet ricas.

2.1 RICH INTERNET APPLICATIONS

Atualmente é possível observar um crescimento na migração de uma multiplicidade de aplicações *desktop* para *web* (POWELL; NAKAMURA; AKAMA, 2009).

A tendência atual é o desenvolvimento de aplicações de negócio desenvolvidas para a Internet utilizando tecnologias relacionadas à *HiperText Markup Languagem* (HTML) (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). Contudo, esses autores ressaltam que HTML foi originalmente criada como uma linguagem de marcação de hipertexto baseada na Internet e para compor páginas *web* estáticas. E, ainda, que HTML não permite a atualização parcial de páginas *web*. Isso se tornou possível com a introdução de programação em JavaScript que pode ser utilizada para manipular elementos específicos da tela (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012).

Essa migração de aplicações como desenvolvimento de tecnologias para melhorar as aplicações *web* estáticas definiu as denominadas *Rich Internet Applications* (RIA) (DUHL, 2003). As RIAs são vislumbradas como sendo a unificação do melhor das aplicações *desktop*, tais como a interface de interação com o usuário e tempo de resposta de interação mais rápido, decorrente da não necessidade de recarregar a página inteira, com o melhor das aplicações *web*, com o uso de *download* progressivo para recuperação de conteúdo (POWELL; NAKAMURA; AKAMA, 2009, p. 1).

As várias metodologias usadas no desenvolvimento de aplicações *web* não são rapidamente transferidas para as RIA devido às características desse tipo de aplicação, tais como (POWELL; NAKAMURA; AKAMA, 2009):

1) Carga de aplicações e camadas de interatividade entre o servidor e o cliente;
2) Minimização de dados transferidos entre servidor e cliente por carregamento parcial da página;
3) Distribuição da computação realizada na página entre o cliente e o servidor;
4) Comunicação assíncrona e concorrente entre cliente e servidor (PRECIADO et al., 2005).

O tipo de comunicação que é o modelo de padrão comportamental usado para aplicações web é inadequado para as RIA uma vez que elas podem exibir comportamento mais rico e flexível, tal como a computação envolvendo somente fragmentos parciais da interface (COMAI; CARUGHI, 2007). Além disso, a multiplicidade de tecnologias requeridas para implementar as aplicações *web* em geral, e as RIAs em particular, (como, por exemplo: JavaScript, HTML e *Cascading Style Sheets* (CSS) para o cliente, *JavaServer Pages* (JSP) e *Java Persistence API* (JPA) para o servidor e *Structured Query Language* (SQL), XQuery para o banco de dados), resulta em diversas dificuldades, tais como problemas de segurança, questões de assincronismo e falhas (*gaps*) de implementação das tecnologias (POWELL; NAKAMURA; AKAMA, 2009). Além disso, há pelo menos quatro categorias de RIAs sem um caminho fácil de tradução entre elas (BOZZON et al., 2006, p. 354):

a) Aplicações *web* baseadas em scripts – a lógica no lado cliente é implementada por meio de linguagens de *script*, tais como JavaScript, e a interface é baseada na combinação de HTML e CSS.
b) Aplicações *web* baseadas em *plugin* – a renderização avançada da tela e processamento de eventos são assegurados pelos *plugins* existentes no navegador que interpretam *eXtensible Markup Language* (XML) e pela existência de programas ou arquivos de mídias de propósitos gerais como (Flash, Flex, Lazlo).
c) Aplicações *web* baseada em navegador – a interação rica é nativamente suportada por alguns navegadores que interpretam linguagens de definição de interface declarativas (como *XML User Interface Language* (XUL)).

d) Aplicações *web* baseadas em *desktop* – tecnologias nas quais as aplicações são baixadas da *web* e executadas fora do navegador (Java Web Start, Window Smart).

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 são as ferramentas, as tecnologias, os ambientes de desenvolvimento e outros utilizados para realizar a modelagem e a implementação do aplicativo. O Quadro 1 apresenta as ferramentas e tecnologias utilizadas.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Cacoo Fluxograma		https://cacoo.com/sample	Desenvolvimento de diagrama de caso de uso e DER.
JQuery	2.1.1	http://jquery.com	Biblioteca de JavaScript.
DBDesigner	2.25	http://www.fabforce.net/dbdesigner4/	Modelagem do banco de dados.
Sublime Text	3	http://www.sublimetext.com/	Ambiente de desenvolvimento.
Apache	7.0	http://www.apache.org/	Servidor *web* para a aplicação.
Bootstrap	3.2.0	http://getbootstrap.com/	Framework para desenvolvimento da interface responsiva.
PostgresSQL	9.4.4	http://www.postgresql.org/	Banco de dados
pgAdmin	1.20.0	http://www.pgadmin.org/	Ferramenta para administração do banco de dados
Wildfly	8.1	http://wildfly.org/	Servidor web para a aplicação
Eclipse Luna	4.4.2	https://www.eclipse.org/	Ambiente de desenvolvimento
JavaServer Faces	2.0	http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html	Framework para desenvolvimento da interface
Java Persistence API	2.2	https://javaserverfaces.java.net/	Para desenvolvimento da interface do aplicativo

Quadro 1 - Ferramentas e tecnologias utilizadas

3.1.1 JavaServer Faces

O JavaServer Faces (JSF) é um *framework* que tem por característica o uso de ManagedBean para o gerenciamento de informações. Os ManagedBeans são objetos fundamentais de uma aplicação JSF, pois fornecem dados que serão exibidos na tela, recebe os dados enviados nas requisições e executam tarefas de acordos com as ações dos usuários.

Cada ManagedBean possui um tipo de escopo, ideal para cada situação. Os principais tipos de escopo são o *request*, *view*, *session* e *application*.

No escopo *request*, as instâncias dos ManagedBean são criadas durante o processamento de uma requisição assim que forem necessárias e descartados no final desse mesmo processamento. Os dados não são mantidos de uma requisição para outra. A partir do JSF 2, foi adotado o escopo *request* como padrão, não sendo necessário explicitar a escolha do escopo *request* por meio da anotação *@RequestScoped*. A Figura 1 representa esquematicamente a atuação do escopo *request*.

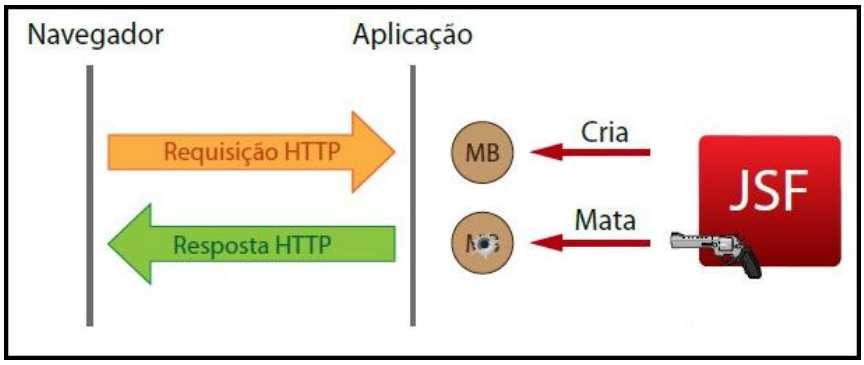

Figura 1 - Escopo *Request.*

O escopo *view* foi adicionado no JSF 2, esse escopo mantém dados enquanto a tela não é mudada. As instâncias dos ManagedBean em escopo view são eliminadas apenas quando há uma navegação entre telas. Para escolher o escopo view, deve-se anotar o ManagedBean com a utilizar a anotação *@ViewScoped*. Na Figura 2 está uma representação esquemática do funcionamento do escopo *view*.

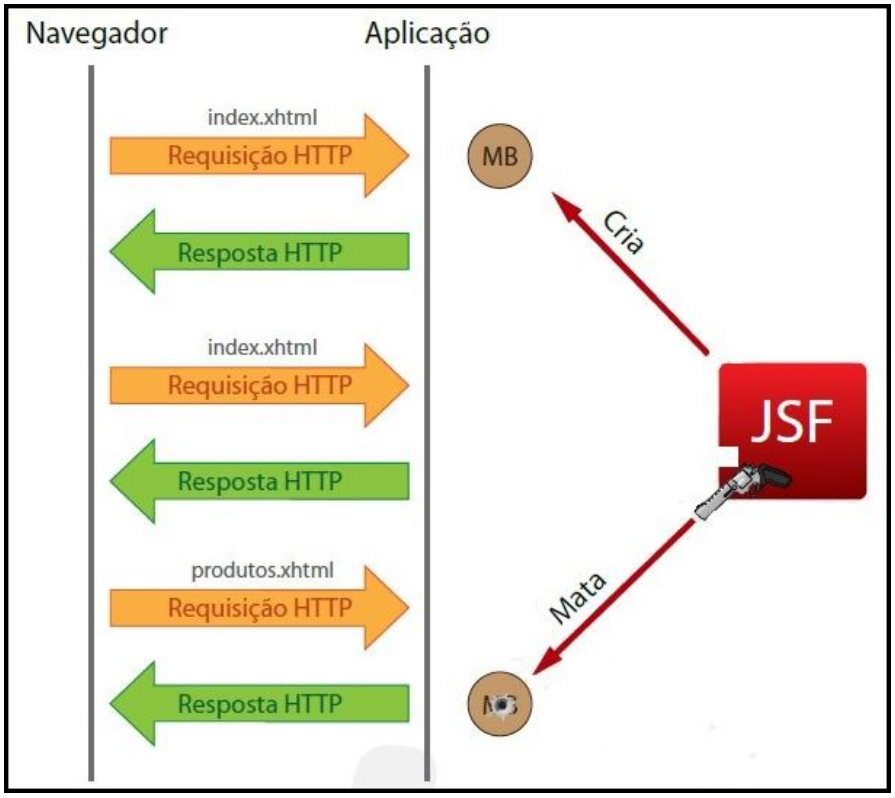

Figura 2 - Escopo *View*.

Para escolher o escopo *session*, deve-se utilizar a anotação *@SessionScoped*. As instâncias do ManagedBean configuradas com o escopo *session* são criadas quando necessárias durante o processamento de uma requisição e armazenadas na *session* do usuário que fez a requisição. Essas instâncias são eliminadas basicamente em duas situações: a aplicação decide por algum motivo especifico excluir a *session* de um usuário, por exemplo, quando o usuário faz *logout*, ou o servidor decide excluir a *session* de um usuário quando esse usuário não faz uma requisição por um determinado período de tempo. A representação do funcionamento do escopo *session* está na Figura 3.

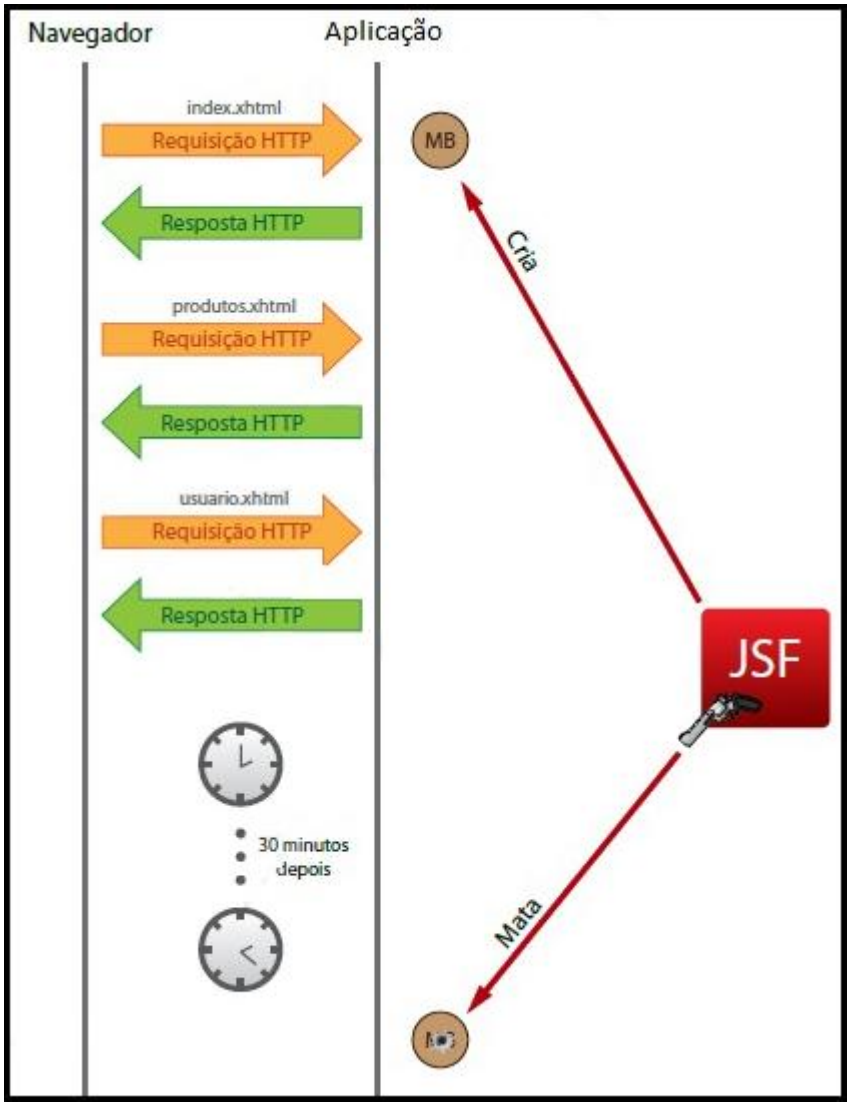

Figura 3 - Escopo *Session.*

Para escolher o escopo *application*, deve-se utilizar a anotação *@ApplicationScoped*. As instâncias dos ManagedBean configurados com escopo *application* são criados no primeiro momento em que elas são utilizadas e mantidas até a aplicação ser finalizada. Ao contrário dos outros escopos, as instâncias dos ManagedBean registrados com escopo *application* são compartilhados com todos os usuários da aplicação. O JSF cria apenas uma instância de cada ManagedBean em escopo de *application*.

3.1.2 Java Persistence API

A especificação *Java Persistence API* foi criada com objetivo de padronizar as ferramentas *Object-Relational Mapping (*ORM) para aplicações Java e consequentemente diminuir a complexidade do desenvolvimento (K19, 2012).

Ferramentas ORM são ferramentas utilizadas para automatizar a transição de dados entre as aplicações e os Sistemas de Gerenciamento de Banco de Dados (SGBD), facilitando assim a comunicação entre aplicações Java que seguem o modelo orientado a objetos e os SGBDs que seguem o modelo relacional. A principal ferramenta ORM para Java utilizada no mercado é o Hibernate (K19, 2012).

A JPA é apenas uma especificação, não implementa nenhum código. A JPA especifica um conjunto de classes e métodos que as ferramentas ORM devem implementar.

Para configurar o Hibernate em uma aplicação, é necessário criar um arquivo chamado persistence.xml. O conteúdo desse arquivo possuirá informações sobre o banco de dados, como a *Uniform Resource Locator* (URL) de conexão, usuário e senha, além de dados sobre a implementação de JPA que será utilizada.

Principais anotações Java de mapeamento do JPA:

- *@Entity* - as classes anotadas com *@Entity* são mapeadas para tabelas, por convenção, as tabelas possuem o mesmo nome que as classes, mas podem ser alteradas utilizando a anotação *@Table*. Os atributos declarados nessas classes são mapeados para colunas na tabela correspondente a classe, por convenção, as colunas possuem o mesmo nome dos atributos, podendo ser alterado o nome com a anotação *@Column.*
- *@Id* - define que um atributo da classe anotada com @Entity será mapeado para a chave primaria da tabela.
- *@GeneratedValue* - geralmente declarado juntamente com a anotação *@Entity*. Define que o valor do atributo que compõem a chave primaria deve ser gerado pelo banco de dados no momento em que é inserido um novo registro.

Uma das vantagens de utilizar uma implementação JPA é sua capacidade de gerar as tabelas no banco de dados. A criação das tabelas é realizada de acordo com as anotações colocadas nas classes e as informações presentes no arquivo persistence.xml.

3.2 MÉTODO

A seguir estão as fases ou etapas que definem as principais atividades realizadas para o desenvolvimento do trabalho. Essas etapas são definidas de acordo com o proposto pelo modelo sequencial linear definido em Pressman (2006).

a) Levantamento de requisitos

O levantamento dos requisitos foi realizado a partir de uma conversa com a enfermeira responsável pelo setor médico-odontológico da UTFPR, Campus Pato Branco. Essa enfermeira expôs as atuais dificuldades com a forma atual de registro dos atendimentos, os procedimentos de trabalho realizados no setor e as funcionalidades desejadas para um aplicativo para ser utilizado no referido setor. No levantamento de requisitos foi identificado que não há necessidade de usuários distintos no sistema, no sentido de restrição de funcionalidades. E também que a principal funcionalidade do sistema, a sua regra essencial de negócio, está centrada no registro dos atendimentos e que esse registro seja classificado por tipo de sintoma, doença ou outro.

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 como um diagrama de entidades e relacionamentos do banco de dados e por meio do esboço da tela inicial e da tela principal do sistema.

c) Implementação do sistema

Para a implementação do sistema foram utilizadas as tecnologias *Java Enterprise Edition* (Java EE), como *Enterprise Java Bean* (EJB), para fazer o encapsulamento da lógica de negocio da aplicação e JPA (*Java Persistence API*), para o gerenciamento dos dados, utilizando mapeamento relacional de objeto (*Object-Relational Mapping* (ORM)). Para a implementação da parte Web, foi utilizada o framework JSF, que é especificação baseada em componentes para a construção de interfaces de usuários, Ajax e JavaScript, utilizados principalmente na parte cliente da aplicação, permitindo que sejam feitas alterações na pagina sem a necessidade de atualizar todos o seu conteúdo.

d) Realização dos testes

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

e) Implantação do sistema

O sistema foi implantado para uso em um servidor da própria Universidade.

4 RESULTADOS

Este capítulo apresenta o que foi obtido como resultado da realização deste trabalho. Inicialmente é apresentado o escopo do sistema, em seguida a modelagem do problema e da solução. O sistema desenvolvimento é apresentado por meio das suas telas e a codificação realizada por listagens de código explicadas.

4.1 ESCOPO DO SISTEMA

O sistema de atendimento médico e odontológico foi desenvolvido de controlar e manter um histórico dos atendimentos e encaminhamentos realizados na UTFPR, Câmpus Pato Branco, bem como o controle de estoque dos medicamentos.

Com o sistema para gerenciamento de atendimento do setor médico - odontológico, será possível conseguir mapear de forma digital todos os atendimentos e encaminhamentos realizados, possibilitando a geração de relatórios. Dessa maneira, será evitado o armazenamento de informações em papel e também o controle de forma manual. Garantindo assim a segurança desses dados, que poderão ser acessados por qualquer computador da entidade, mediante identificação por meio de um controle de acesso com login e senha.

Com o controle de estoque dos medicamentos será possível acompanhar e de forma mais simples as variações no estoque, já que este se atualiza a cada baixa de medicamento, o que garante que não se tenham medicamentos em falta ou em excesso.

4.2 MODELAGEM DO SISTEMA

Na Figura 4 é exibido o diagrama de casos de uso do sistema. Nele é documentado o que o sistema faz no ponto de vista do usuário, descrevendo as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema.

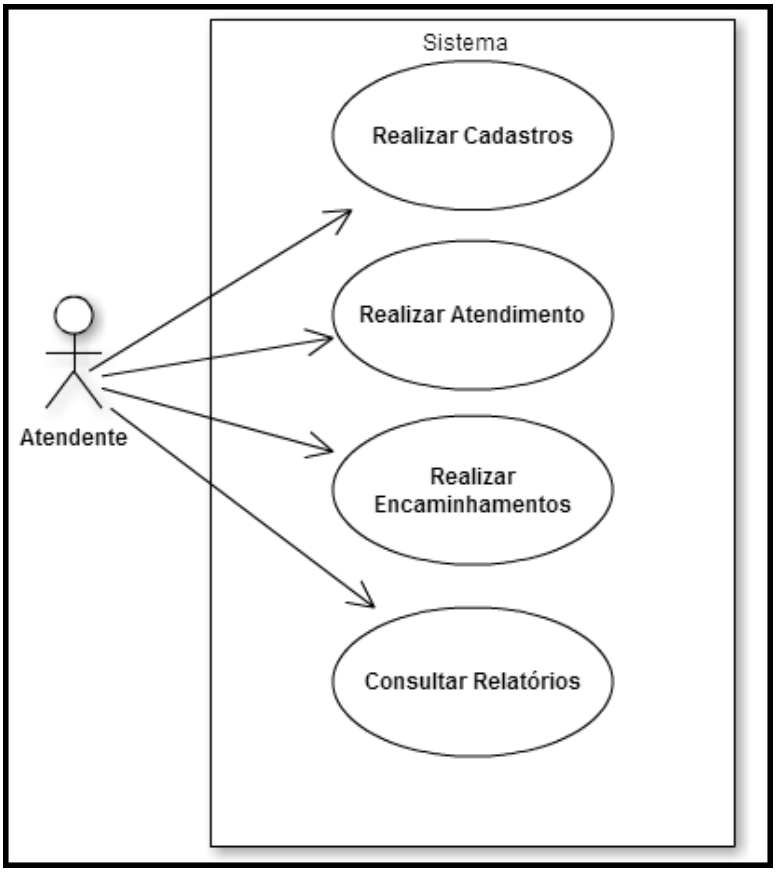

Figura 4 - Diagrama de Caso de Uso

Como representado na Figura 4, diagramas de casos de uso são compostos basicamente por quatro partes:

1. Cenário: Sequência de eventos que acontecem quando um usuário interage com o sistema.
2. Ator: Usuário do sistema, ou melhor, um tipo de usuário.
3. Use Case: É uma tarefa ou uma funcionalidade realizada pelo ator (usuário)
4. Comunicação: é o que liga um ator com um caso de uso

O Quadro 2 apresenta a listagem dos requisitos funcionais identificados para o sistema.

Identi-ficação	Nome	Descrição
01	Cadastrar Curso	Manter os cursos no sistema.
02	Cadastrar Departamento	Manter os departamentos no sistema.
03	Cadastrar Doença	Manter os departamentos no sistema.
04	Cadastrar Encaminhamento	Manter os encaminhamentos no sistema.
05	Cadastrar Especialidade	Manter as especialidades no sistema.

06	Cadastrar Medicamento	Manter os medicamentos no sistema.
07	Cadastrar Profissional	Manter os profissionais no sistema.
08	Cadastrar Tipo de paciente	Manter os tipos de paciente no sistema.
09	Cadastrar Unidade	Manter os tipos de unidade no sistema.
10	Cadastrar Usuário	Manter os tipos de usuário no sistema.

Quadro 2 - Requisitos Funcionais

A Figura 5 apresenta o diagrama de entidade de relacionamento, mostrando o relacionamento entre as tabelas.

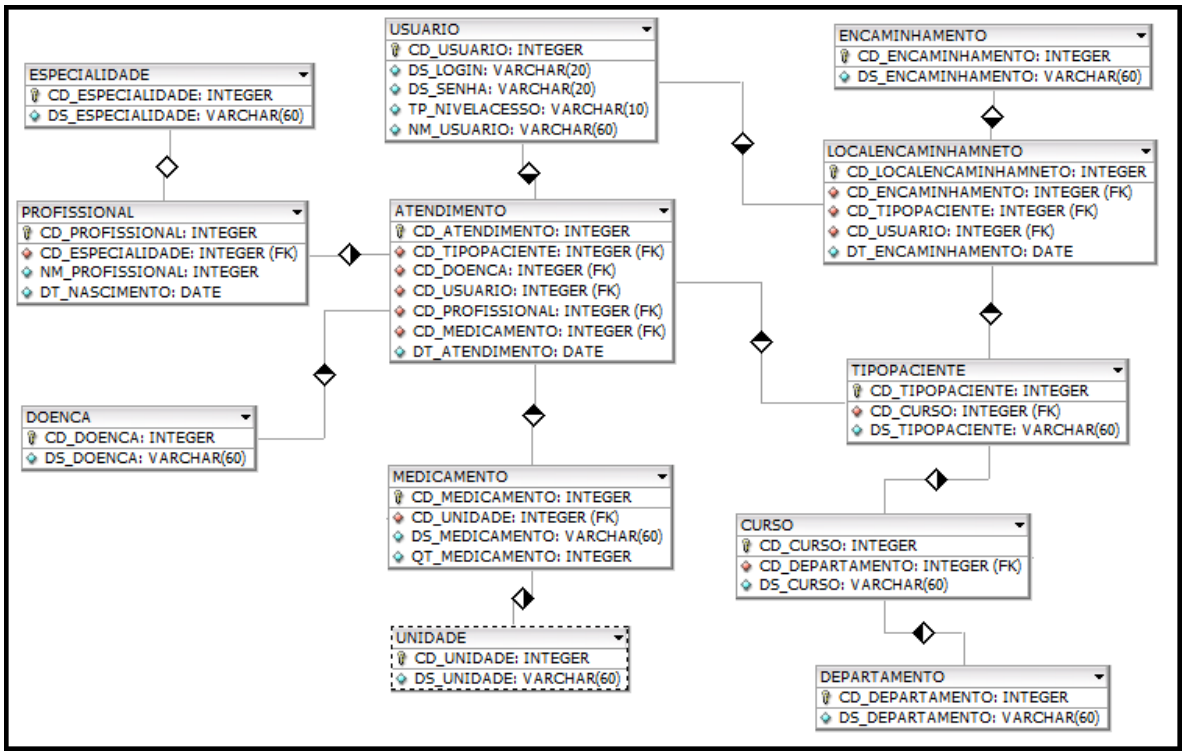

Figura 5 - Diagrama de Entidade Relacionamento

A parte principal do sistema é o atendimento médico e odontológico. Essa entidade possui um relacionamento de um para um com profissional, doença, medicamento, tipo de paciente e usuário.

Ao ser feito um atendimento é informado o profissional que realizou o atendimento, qual o tipo de paciente está sendo atendido, qual a doença diagnosticada, o medicamento e a quantidade indicada.

Outra parte principal do sistema são os encaminhamentos a outros locais de atendimentos que são necessários devido à gravidade das ocorrências. Os encaminhamentos são feitos, por exemplo, à UPA, para que seja realizado o atendimento adequado.

Ao realizar um encaminhamento é informado o local que está sendo encaminhado e o tipo de paciente.

O profissional tem um relacionamento de um para muitos com especialidade. O medicamento tem um relacionamento de um para um com a unidade. O tipo de paciente tem um relacionamento de um para um com curso, que por sua vez tem um relacionamento de um para um com departamento.

4.3 APRESENTAÇÃO DO SISTEMA

O sistema possui um menu com as funcionalidades de cadastro, atendimento, encaminhamento e relatórios.

4.3.1 Cadastros

No menu cadastro é possível acessar todas as telas de cadastro do sistema, sendo elas: Curso, Departamento, Doença, Encaminhamento, Especialidade, Medicamento, Profissional, Tipo de Paciente, Unidade e Usuário. A Figura 6 mostra o menu de cadastro.

Figura 6 - Menu de cadastros

No cadastro de curso são informados os cursos que a instituição oferece. A tela de cadastro de curso possui três botões para realizar as ações de cadastrar um novo curso, editar o curso e excluir o curso. Logo abaixo dos botões é apresentada uma lista com os cursos já cadastrados. A Figura 7 mostra a tela de listagem de curso. Ao selecionar a opção de um novo curso, será aberta uma tela para o cadastro de um novo curso.

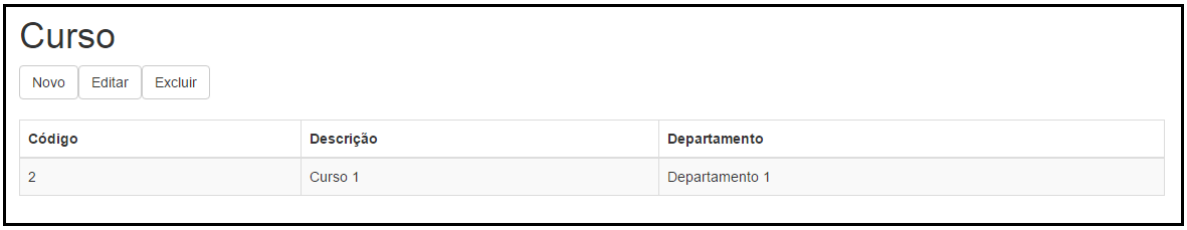

Figura 7 - Listagem de Curso

A Figura 8 exibe a tela de adição e alteração. Para editar ou excluir um curso é necessário que seja selecionado um curso na lista exibida na Figura 7.

Figura 8 - Cadastro de Curso

Na Figura 9 é exibido o cadastro de departamentos da instituição, os quais são responsáveis pela coordenação dos cursos. Assim como na tela de cadastro de cursos, a tela de cadastro de departamento possui três botões para realizar as ações de cadastrar um novo departamento, editar ou excluir um departamento, e abaixo é exibida uma lista com os departamentos já cadastrados.

Figura 9 - Cadastro de Departamento

Para editar ou excluir um departamento é necessário que este seja selecionado um registro na listagem de departamento, conforme exibida na Figura 10.

Figura 10 - Listagem de Departamento

Todos os cadastros seguem o mesmo padrão de leiaute, com as opções de edição, alteração e exclusão acima da listagem das informações já gravadas no banco de dados. E a tela de cadastro que apresenta os campos para serem inseridas as informações e as opções de gravar ou de cancelar a alteração feita no registro. Em alguns casos, as telas de cadastro dependem de outras informações já gravadas, por exemplo, medicamento depende de unidade, encaminhamento depende de um profissional, de medicamento, de tipo de paciente e de usuário.

No cadastro de usuário são inseridos os usuários que farão a utilização do sistema para a realização dos cadastros ou para fazer o registro dos atendimentos. A tela de listagem de usuários segue o mesmo padrão das demais telas, com os botões para realizar as ações de cadastrar, edição e exclusão excluir um usuário, logo abaixo dos botões é mostrado um grid com os usuários já cadastrados. A

Figura 11 mostra a tela de cadastro de usuário. Ao selecionar a opção de um novo usuário, será incluso um novo usuário.

Figura 11 - Listagem de Usuário

4.3.2 Atendimento

Na Figura 12 é exibido o menu para acesso ao atendimento dos pacientes. O atendimento pode ser médico ou odontológico. Esta é a tela principal do sistema.

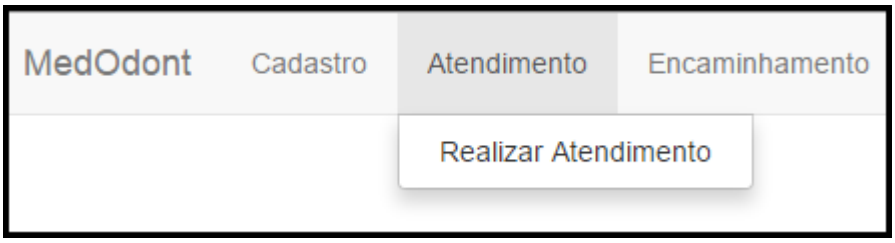

Figura 12 - Menu de acesso ao Atendimento

Os atendimentos são realizados no momento em que um paciente chega ao local de atendimento médico ou odontológico. Como são situações que não podem ser agendas, não teve a necessidade de se ter um agendamentos dos atendimentos. Assim, o paciente chega e já é atendido.

A Figura 13 mostra a tela de atendimentos realizados. O usuário pode fazer o lançamento de um novo atendimento, editar ou excluir um atendimento já gravado.

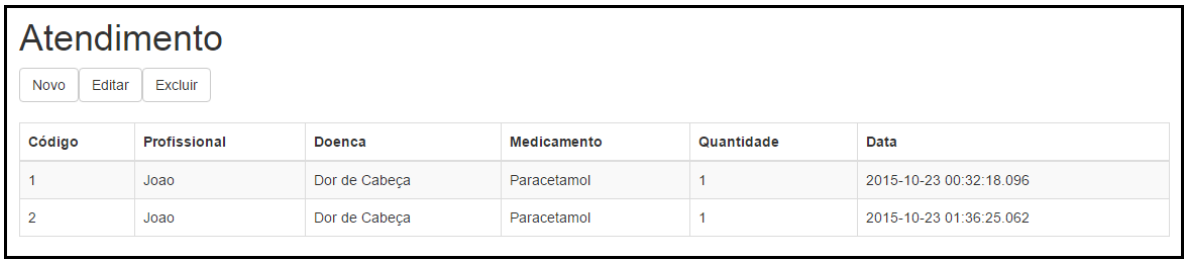

Figura 13 - Listagem de Atendimentos

Ao selecionar um novo atendimento, deve ser informado o profossional que atendeu o paciente, qual o tipo do paciente, qual a doença que o paciente possui, o

medicamento utilizado e a quantidade recomendada. A Figura 14 mostra a tela de cadastro de um novo atendimento.

Figura 14 - Cadastro de Atendimento

Ao informar a quantidade do medicamento, o sistema validará se existe estoque do medicamento selecionado. Caso exista será diminuído a quantidade receitada ao paciente do estoque. Caso não haja estoque suficiente, será mostrada uma mensagem ao usuário, informando-o que não existe a quantidade em estoque do medicamento.

4.3.3 Encaminhamento

A Figura 15 mostra o acesso a tela de encaminhamentos. Os encaminhamentos são realizados para pacientes que não há condição de atendimento devido ao tipo ou gravidade de enfermidade.

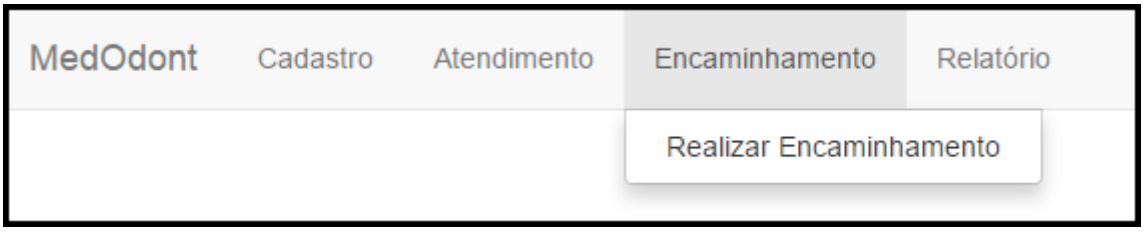

Figura 15 - Menu acesso Encaminhamento

A Figura 16 mostra a tela de encaminhamentos já realizados. O usuario pode fazer o lançamento de um novo encaminhamento, editar ou excluir um encaminhamento já gravado.

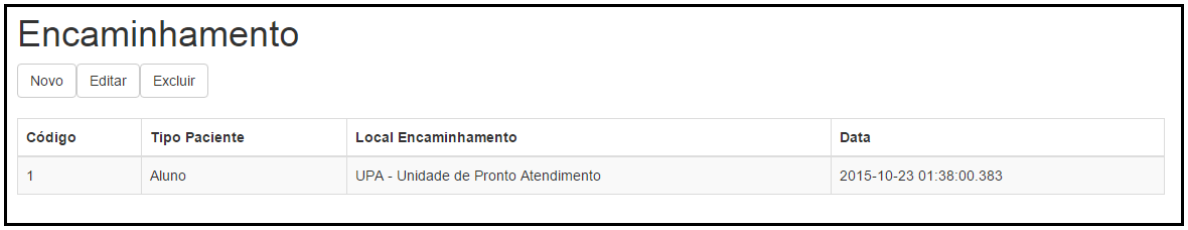

Figura 16 - Listagem de encaminhamentos realizados

Ao selecionar um novo encaminhamento, é informado o tipo do paciente e qual o destino de atendimento deste paciente. Ao salvar o encaminhamento, o sistema grava a data do registro e qual o usuario que fez esse processo. A Figura 17 mostra a tela de registro de um novo encaminhamento, que é feito quando o atendimento não pode ser realizado no setor ou pelos profissionais do setor.

Figura 17 - Registro de Encaminhamento

4.4 IMPLEMENTAÇÃO DO SISTEMA

O sistema foi desenvolvido utilizando as tecnologias Java EE, JSF, JPA, Hibernate, JavaScript e CSS. Foi utilizado a IDE Eclipse Luna para a programação, o servidor web utilizado foi o Wildfly 8.1, o banco de dados é Postgres 9.4.4.

A Listagem 1 mostra a configuração do Hibernate, criado em um arquivo chamado persistence.xml, o qual deve estar localizado na pasta META-INF do

projeto. O conteúdo deste arquivo possui informações sobre o banco de dados, como URL de conexão, usuário, senha e dados sobre a implementação da JPA.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
<persistence-unit name="PostgreSQL" transaction-type="JTA">
        <jta-data-source>java:jboss/datasources/TCCJava</jta-data-source>
        <class>br.edu.utfpr.model.Atendimento</class>
        <class>br.edu.utfpr.model.Curso</class>
        <class>br.edu.utfpr.model.Departamento</class>
        <class>br.edu.utfpr.model.Doenca</class>
        <class>br.edu.utfpr.model.Encaminhamento</class>
        <class>br.edu.utfpr.model.Especialidade</class>
        <class>br.edu.utfpr.model.LocalEncaminhamento</class>
        <class>br.edu.utfpr.model.Medicamento</class>
        <class>br.edu.utfpr.model.Profissional</class>
        <class>br.edu.utfpr.model.TipoPaciente</class>
        <class>br.edu.utfpr.model.Unidade</class>
        <class>br.edu.utfpr.model.Usuario</class>
        <properties>
            <property name="hibernate.transaction.jta.platform"
                value="org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

Listagem 1 - Arquivo persistence.xml

Na Listagem 2 é exibido o arquivo web.xml que contém as informações para realizar os ajustes dos parâmetros da aplicação, como mapeamento, filtros, segurança, conversores, entre outros.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
    <display-name>PosJava</display-name>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <welcome-file-list>
        <welcome-file>utfpr/index.xhtml</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
</web-app>
```

Listagem 2 - Arquivo web.xml

Visando facilitar e agilizar o desenvolvimento foi criado um *template* para as páginas web do sistema. A ideia dos *templates* é identificar um padrão nas telas e defini-los por meio de esqueleto. As importações das classes JSF são as mesmas utilizadas em todas as telas do sistema.

A Listagem 3 mostra o *template* utilizado no sistema, já com as importações das classes JSF, dos arquivos JavaScripts e dos arquivos CSS. No *template*, os componentes <h:head> e <h:body> foram definidos estaticamente. Assim, todas as telas que utilizam esse *template* não correm o risco de ficarem sem esses componentes.

A utilização das *tags* <h:head> e <h:body> é fundamental para o funcionamento completo das páginas HTML geradas pelo JSF. Ao processar essas *tags*, na etapa Render Response, o JSF adiciona recursos como scripts e arquivos de estilo na tela HTML que será enviada para o usuário. Esses recursos são necessários para o funcionamento correto dos componentes.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:h="http://java.sun.com/jsf/html"
6      xmlns:ui="http://java.sun.com/jsf/facelets">
7  <h:head>
8      <h:outputStylesheet name="common_style.css" library="css" />
9      <h:outputStylesheet name="bootstrap.min.css" library="css" />
10     <h:outputStylesheet name="navbar.css" library="css" />
11     <h:outputScript library="js" name="jquery.min.js" />
12     <h:outputScript library="js" name="bootstrap.min.js" />
13     <h:outputScript library="js" name="ie10-viewport-bug-workaround.js" />
14     <title>Sistema de Atendimento MedOdont</title>
15 </h:head>
16 <h:body>
17     <div id="page">
18         <div id="header">
19             <ui:insert name="header">
20                 <ui:include src="header.xhtml" />
21             </ui:insert>
22         </div>
23         <h:form prependId="false">
24             <div id="content">
25                 <ui:insert name="content">
26                     <ui:include src="content.xhtml" />
27                 </ui:insert>
28             </div>
29         </h:form>
30     </div>
31 </h:body>
32 </html>
```

Listagem 3 - Template do sistema

Para criar as demais telas do sistema, as quais utilizam um determinado *template*, deve-se criar um arquivo XHTML e adicionar a *tag* <ui:composition> à esse arquivo. Todo conteúdo fora da *tag* <ui:composition> será descartado pelo JSF no processo de construção da tela. Se o conteúdo de um trecho dinâmico não for definido, o JSF utilizará o conteúdo existente no corpo da *tag* <ui:insert> definido no *template*.

A Listagem 4 mostra uma classe Java anotada com *@Entity* sinalizando que esta classe deve ser mapeada pela implementação do JPA. Essa entidade corresponde a uma do banco de dados. As anotações *@Table* e *@Column* são utilizadas para personalizar o nome das tabelas e das colunas.

A chave primária da tabela é definida pela anotação *@Id*, quando seguido da anotação *@GeneratedValue*, o banco de dados fica responsável pela geração do valor da chave primária.

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

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

    @Column(name = "DS_UNIDADE", length = 60, nullable = false)
    private String descricao;

    public Unidade() {
    }

    public Unidade(int codigo, String descricao) {
        this.codigo = codigo;
        this.descricao = descricao;
    }
```

Listagem 4 - Classe Java: Unidade.java

Na Listagem 5 são apresentados os componentes da tela de listagem das unidades. Nesta listagem pode-se observar vários componentes JSF. Dentre eles temos o componente h:outputLink, utilizado para criar o botão Novo, que fornece um *link* para a tela de adição de uma nova unidade. O componente h:commandButton, utilizado para criar o botão editar e excluir, que são ligados a um ManagedBean para executar suas respectivas ações.

```
...
<div class="container">
    <div class="starter-template">
        <h1>Unidade</h1>
    </div>
    <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"
        actionListener="#{pesquisaUnidadeBean.editar()}"/>
    <h:commandButton styleClass="btn btn-default" value="Excluir" id="excluir"
        actionListener="#{pesquisaUnidadeBean.excluir()}"/>
    <div id="divErro" class="alert alert-danger" style="display: none;"/>
    <div class="spacer">
        <br />
    </div>
    <h:inputHidden value="#{pesquisaUnidadeBean.codigo}" id="pk" />
    <h:dataTable id="tabela" value="#{pesquisaUnidadeBean.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="Descrição"/>
            </f:facet>
            <h:outputText value="#{item.descricao}"/>
        </h:column>
    </h:dataTable>
    <div class="spacer" />
    <h:outputScript library="js" name="selecaoCodigo.js" />
</div>
...
```

Listagem 5 - Tela de listagem das unidades

Na Listagem 6 é apresentado o código fonte da tela de cadastro e alteração de unidade. O componente <h:inputText> está vinculado à propriedade descrição do ManagedBean ListaUnidadeBean.

```
...
<h:form>
    <div class="container">
        <h1>Unidade</h1>
        <div class="spacer">
            <br />
        </div>
        <hr/>
        <div class="form-group">
            <div class="row">
                <div class="col-md-6">
                    <h:outputLabel for="descricao" value="Descrição" class="control-label"/>
                    <h:inputText value="#{listaUnidadeBean.unidade.descricao}"
                    id="descricao" class="form-control" required="true"
                    pt:placeholder="Informe aqui a unidade"
                    validatorMessage="Descrição inválida"
                    requiredMessage="É necessário informar uma descrição"/>
                    <h:message id="m_descricao" for="descricao"/>
                </div>
            </div>
        </div>
        <hr/>
        <div class="form-group">
            <div class="row">
                <div class="col-md-12">
                    <h:commandButton value="Salvar" actionListener="#{listaUnidadeBean.salvar()}"
                        class="btn btn-default"/>
                    <h:outputLink styleClass="btn btn-default" value="cadastro.xhtml">
                        <h:outputText value="Cancelar"/>
                    </h:outputLink>
                </div>
            </div>
        </div>
    </div>
</h:form>
...
```

Listagem 6 - Código fonte da tela de adição/alteração de unidade

Os ManagedBean são objetos fundamentais de uma aplicação JSF, pois fornecem dados que serão exibidos na tela. Recebem os dados enviados nas requisições e executam tarefas de acordo com as ações dos usuários.

Um ManagedBean pode ser definido por meio da criação de uma classe Java, Listagem 7, e declarar esta classe no arquivo faces-config.xml, Listagem 8, definindo o nome, a classe e o escopo do ManagedBean.

TesteBean.java

```
1  package br.edu.utfpr.bean;
2
3  public class TesteBean {
4
5  }
6
```

Listagem 7 - Classe Java

```
<managed -bean>
   <managed -bean-name>testeBean</managed -bean-name>
   <managed -bean-class>br.edu.utf.bean.TesteBean</managed -bean-class>
   <managed -bean-scope>request</managed -bean-scope>
</managed -bean>
```

Listagem 8 - Arquivo faces-config.xml

Nesta aplicação foi utilizada uma segunda maneira de se declarar um ManagedBean. Foi utilizada a anotação @ManagedBean, com isso o JSF assumirá que o nome do ManagedBean é o nome da classe com a primeira letra em minúsculo. Além disso, por padrão o escopo deste ManagedBean é *request*.

Para acessar ou modificar os valores dos atributos do ManagedBean em uma tela JSF, são necessários os métodos *getters* e *setters* implementados na classe anotada como ManagedBean. Para exibir ou modificar o valor do atributo, esse atributo deve estar vinculado a um componente JSF na tela utilizando a expressão de linguagem (#{}).

O vínculo com uma propriedade de um ManagedBean dá-se por meios dos métodos *getters* e *setters* e não pelo nome do atributo.

Na Listagem 9 está uma classe com a anotação *@ManegedBean*, responsável pelo cadastro de unidade. Este ManagedBean teve seu escopo alterado para *view* com a anotação *@ViewScoped*, com isso, os dados são mantidos enquanto o usuário não mudar de tela.

```
@ManagedBean(name="listaUnidadeBean")
@ViewScoped
public class ListaUnidadeBean extends asbtractCadastro implements Serializable{
    private static final long serialVersionUID = 1L;

    private Unidade unidade;

    @Inject
    private UnidadeController controller;

    @PostConstruct
    private void inicialiar(){
        setUnidade(new Unidade());
        if (getCodigo() != null){
            unidade = controller.pesquisarPorCodigo(getCodigo());
        }
    }

    @Override
    public void cancelar() throws IOException {
        setUnidade(new Unidade());
        redirect("/utfpr/unidade/cadastro.xhtml");
    }

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

Listagem 9 - ManagedBean responsável pelo cadastro da Unidade

No ManagedBean da Listagem 9 pode-se visualizar o método *inicializar()* anotado com *@PostConstruct*. Para que este método seja executado após ser criado o ManagedBean. O método *inicializar()* cria uma nova instância de unidade,

ao começar a inserção de uma nova unidade. Caso seja selecionado uma unidade já gravada para ser altera é feito uma pesquisa pelo código da unidade.

Todas as telas de cadastro possuem as ações de salvar e cancelar a alteração que o usuário está realizando. Visando melhorar a manutenção do código fonte foi criado uma classe abstrata e declarado os métodos de salvar e cancelar. No ManagedBean da Listagem 9 foi estendida esta classe abstrata e implementado os métodos *salvar()* e *cancelar()*.

O método salvar executa o comando de alterar do controlador da unidade, passando por parâmetro um objeto do tipo Unidade para ser persistido no banco de dados, em seguida redireciona para a listagem das unidades. O método cancelar apenas instancia uma nova unidade e redireciona para a página de cadastro para que os dados digitados sejam apagados.

Na Listagem 10 é apresentado o ManagedBean responsável por listar as unidades cadastradas. Possui o método *inicializar()* anotado com *@PostContruct* para que seja executado imediatamente após a construção do ManagedBean e carregue uma lista com as unidades já cadastradas.

Da mesma forma que o ManagedBean da Listagem 9, o ManagedBean da Listagem 10, estende uma classe abstrata que tem as ações de editar e excluir uma unidade. O método editar redireciona para a tela de cadastro e excluir executa o método excluir do controlador da Unidade, exclui a unidade e redireciona para a página de listagem das unidades.

```java
@ManagedBean(name = "pesquisaUnidadeBean")
@ViewScoped
public class PesquisaUnidadeBean extends abstractLista implements Serializable{
    private static final long serialVersionUID = 1L;
    private Unidade unidade;
    private List<Unidade> lista;

    @Inject
    private UnidadeController controller;

    @PostConstruct
    private void inicializar() {
        setLista(controller.listar());
    }

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

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

Listagem 10 - ManagedBean responsável pela listagem de Unidade

5 CONCLUSÃO

O objetivo de desenvolver um sistema web para o registro dos atendimentos realizados no setor médico-odontológico da UTFPR, Câmpus Pato Branco, foi alcançado, visto que esta aplicação facilitará o trabalho e o registro dos atendimentos e encaminhamentos realizados neste setor, bem como, ter um melhor controle dos estoques de medicamentos utilizados.

A aplicação será utilizada por vários usuários distintos, optou-se por um sistema *web*, já que é de fácil entendimento e utilização, o que vem a facilitar os registros propostos. O sistema *web* apresentado foi implementado na linguagem Java e utilizou-se de diversas ferramentas, as quais contribuíram no processo de desenvolvimento.

O desenvolvimento da aplicação possibilitou o uso de conteúdo de todas as disciplinas ofertadas durantes o curso, as quais contribuíram para o desenvolvimento do sistema, bem como para demandas futuras que poderão ser executadas.

Para dar sequência a esta aplicação, futuramente podem ser implementadas novas funcionalidades para que a utilização desta possa trazer maiores benefícios e informações, tais como, geração de relatórios, melhorias nos cadastros e no processo de atendimento e encaminhamento.

REFERÊNCIAS

COMAI, Sara; CARUGHI, Giovanni Toffetti. **A behavioral model for Rich Internet Applications**. In: Web Engineering, LNCS, v.4607, 2007, p.364-369.

DUHL, Joshua. **White paper: Rich Internet Applications**. Technical report, IDC, November 2003, p. 1-33.

K19, **Persistência com JPA 2 e Hibernate**. 2012. Disponível em: <http://www.k19.com.br/downloads/apostilas/java/k19-k21-persistencia-com-jpa2-e-hibernate>. Acesso em: 19 out. 2015.

PAVLIĆ, Daniel; PAVLIĆ, Mile; JOVANOVIĆ, Vladan. **Future of Internet Technologies**. 2012, p. 1366-1371.

POWELL, Courtney; NAKAMURA, Keisuke; AKAMA, Kiyoshi. **Towards a formal behavioral model for Rich Internet Applications.** 2009, p. 1-5.

PRECIADO, Juan Carlos; 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.