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

FÁBIO BRINGHENTI

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA WEB DE GESTÃO PARA CLÍNICAS DE SAÚDE

Jorge Luis Boeira Bavaresco
Carmen Vera Scorsatto

PASSO FUNDO, 2015

FÁBIO BRINGHENTI

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA WEB DE GESTÃO PARA CLÍNICAS DE SAÚDE

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Jorge Luis Boeira Bavaresco
Co-orientadora: Carmen Vera Scorsatto

PASSO FUNDO, 2015

FÁBIO BRINGHENTI

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA WEB DE GESTÃO PARA CLÍNICAS DE SAÚDE

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

Banca Examinadora:

_____

Nome do Professor(a) Orientador(a)

_____

Nome do Professor(a) Convidado(a)

_____

Nome do Professor(a) Convidado(a)

_____

Coordenação do Curso

PASSO FUNDO, 2015

DEDICATÓRIA

Aos meus pais,
pelo amor e apoio em todas as escolhas que fiz.
À minha esposa,
pelo amor e compreensão nos momentos de ausência.
Aos professores,
pela dedicação em ensinar seus conhecimentos e suas experiências profissionais.

AGRADECIMENTOS

Agradeço a minha mãe, Deise Silvana Antunes Lang, e ao meu pai, Joelson Carlos Piazza Bringhenti, que me apoiaram sempre, independentemente das escolhas que fiz, e nunca me deixaram abandonar meus sonhos.

Agradeço a minha esposa, Naiara Carolina Bischoff Pereira Bringhenti, por contribuir com seu amor, carinho, compreensão e conselhos para que o foco fosse mantido e o objetivo fosse alcançado.

Agradeço à professora Carmen Vera Scorsatto que, embora não soubesse exatamente onde eu queria chegar, soube aparar as arestas até que as ideias fossem modeladas e evoluíssem para um projeto.

Agradeço ao professor Jorge Luis Boeira Bavaresco que com seus ensinamentos e sua experiência profissional contribuiu para o refinamento e conclusão deste trabalho.

Agradeço aos demais professores e funcionários do IFSUL por proporcionarem condições para o ensino e o aprendizado de qualidade que nos é ofertado.

EPÍGRAFE

“A simplicidade é o último grau de sofisticação.”

Leonardo da Vinci

RESUMO

O aumento da demanda por serviços de saúde gerado com o crescimento populacional desafia a gestão das clínicas e consultórios médicos. Uma empresa que faz uso de um sistema de informação adquire vantagens na disputa de mercado. Esta pesquisa destina-se à análise e ao desenvolvimento de um sistema web de gestão para clínicas de saúde, especialmente, para atender a uma clínica de fisioterapia na cidade de Passo Fundo. O software desenvolvido utiliza a plataforma Java EE, JPA com o servidor de persistência Hibernate, JavaSever Faces e banco de dados objeto relacional PostgreSQL.

Palavras-chave: Java EE; análise e desenvolvimento; sistemas web.

ABSTRACT

Increased demand for health services generated by population growth challenges the management of clinics and doctors's offices. A company that makes use of an information system takes advantages in market dispute. This research is intended to analysis and development of a web management system for health clinics, especially designed to attend a physiotherapy clinic in the city of Passo Fundo. Software developed using the Java EE plataform, JPA with persistence server Hibernate, JavaServer Faces and PostgreSQL objetc-relational database.

Key words: Java EE; analysis and development; web systems.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A evolução no tratamento de saúde percebida nos últimos anos deve-se, principalmente, aos avanços nas áreas de computação e de eletrônica. Hoje, com exames computadorizados os resultados são mais precisos e proporcionam aos profissionais da saúde, tanto médicos quanto dentistas, fisioterapeutas, psicólogos, maior confiança nas suas decisões.

Além disso, na conjectura atual da saúde no Brasil, uma clínica ou consultório médico, para ser competitiva, deve dispor de mais de uma forma de marcação de consultas para atrair clientes. Pois, o crescimento populacional dificulta a gestão de consultas em consultórios e clínicas, e um sistema de informações pode prover vantagens do ponto de vista competitivo.

Um software de gestão facilitará a administração das clínicas garantindo aos clientes o acesso aos bens e serviços de saúde, de forma que os profissionais da área terão de ocupar-se apenas com a manutenção e recuperação da saúde dos indivíduos, pois, de acordo com Pressman:

> O software distribui o produto mais importante de nossa era – a informação. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis num determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas. (2011, p.31)

Portanto, o sistema proposto neste estudo, com o uso de conceitos de orientação a objetos, linguagem Java para Web e sistema de banco de dados objeto relacional PostgreSQL – escolhidos por serem gratuitos e multiplataforma – será uma ferramenta para auxiliar na gestão dos consultórios e clínicas.

O capítulo 2 do trabalho apresenta um referencial teórico que embasará o leitor com os tópicos considerados essenciais para o desenvolvimento do estudo e da aplicação. O capítulo 3, por sua vez, documenta o desenvolvimento de um sistema web de gestão para clínicas de saúde. O capítulo 4 apresenta as considerações finais resultantes do desenvolvimento do presente estudo.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um sistema de gestão de clínicas de saúde que utilize o paradigma de orientação a objetos, a fim de aperfeiçoar a experiência e a satisfação do paciente na marcação de consultas, além de colaborar para a administração da clínica de saúde.

1.1.2 Objetivos específicos

O trabalho tem os seguintes objetivos específicos:

- Estudar os conceitos da linguagem de programação Java, bem como o paradigma de orientação a objetos, a plataforma Java EE, o padrão de projeto MVC, a API JavaServer Faces, além do uso da JPA com o provedor de persistência Hibernate;
- Examinar o sistema de gerenciamento de banco de dados PostgreSQL, bem como os conceitos de banco de dados relacional;
- Analisar alguns sistemas de gestão de saúde já existentes no mercado: HiDoctor, MedPlus, Medsystem Web e Totvs Saúde;
- Desenvolver a análise do sistema proposto;
- Programar um sistema de gestão web para clínica de fisioterapia utilizando os conceitos estudados;
- Hospedar o sistema em um servidor web, a fim de abstrair problemas de armazenamento de dados e hardware obsoleto ou danificado.

2 REFERENCIAL TEÓRICO

Os assuntos que embasam teoricamente o projeto serão apresentados a seguir. Os tópicos considerados essenciais para o desenvolvimento do estudo e da aplicação e que, por isso, serão apresentados são o paradigma de orientação a objetos e a linguagem de programação Java; a plataforma *Java Enterprise Edition* (Java EE); os componentes *Enterprise Java Beans* (EJB's); a *Aplication Programming Interface* (API) *Java Persistence API* (JPA) e o servidor de persistência *Hibernate*; a API *JavaServer Faces* (JSF); o padrão de projeto *Model-View-Controller* (MVC); os sistemas de banco de dados e o sistema de gerenciamento de banco de dados PostgreSQL. Por fim, serão apresentados alguns sistemas de gestão médica que estão no mercado com suas características.

2.1 O PARADIGMA DE ORIENTAÇÃO A OBJETOS

Bezerra (2007) conceitua o paradigma de orientação a objetos como uma forma de abordar um problema e, ainda, cita o postulado de Alan Kay – um dos pais da orientação a objetos – chamado de "analogia algébrico-biológica", onde ele imagina um sistema de software funcionando do mesmo modo que um organismo vivo. Nele, cada célula interagiria com outras através de mensagens, a fim de realizar um objetivo comum; assim, cada célula se comportaria como uma unidade autônoma.

Esse postulado ditou alguns princípios da orientação a objetos, tais como:

i. Tudo pode ser um objeto;
ii. Os objetos realizam tarefas por meio da requisição de serviços a outros objetos;
iii. Cada objeto pertence a uma determinada classe e uma classe agrupa objetos similares;
iv. A classe é uma caixa preta, na qual comportamentos associados a um objeto são guardados;
v. As classes pertencem a uma hierarquia.

Ainda, conforme Bezerra (2007), o objeto como unidade autônoma, no paradigma de orientação a objetos, contém seus próprios dados, os quais podem ser manipulados por métodos definidos e interagir com outros objetos para executar determinada tarefa. Dessa forma, aproxima-se do mundo real nos métodos de resolução de problemas cotidianos. Ainda

nesse sentido, um sistema de software é visto como um conjunto de objetos relacionados onde cada objeto é responsável por realizar tarefas específicas.

Pode-se concluir, que a orientação a objetos, como técnica para a modelagem de sistemas, diminui a diferença semântica entre a realidade sendo modelada e os modelos construídos (BEZERRA, 2007). Na próxima seção, será apresentada a linguagem de programação Java que pode ser usada para implementar o paradigma de orientação a objetos.

2.2 LINGUAGEM DE PROGRAMAÇÃO JAVA

Em 1991, a Sun Microsystems reconheceu o avanço tecnológico que os dispositivos eletrônicos sofreram com a revolução dos microprocessadores e financiaram um projeto de pesquisa corporativa interna o qual resultou em uma linguagem baseada em C++, inicialmente chamada de Oak por seu criador, James Goesling, em homenagem a uma árvore de carvalho. No entanto, mais tarde, soube-se que esta linguagem – Oak – já existia. Então, em um encontro da equipe da Sun em uma cafeteria sugeriu-se o nome Java, pois era o nome da cidade de origem de um tipo de café importado consumido por eles (DEITEL; DEITEL, 2010).

Para Barnes e Kölling (2004), a linguagem Java é muito popular no meio acadêmico, principalmente, devido aos seguintes fatores:

a) Fornece uma implementação limpa da maioria dos conceitos orientados a objetos, e, assim, serve bem como uma linguagem introdutória;
b) Existe uma infinidade de materiais bons – livros, tutoriais, sites, exercícios, fóruns –disponíveis gratuitamente e on-line;
c) Comercialmente, tornou-se uma linguagem de programação importante.

Portanto, estes requisitos fazem de Java uma ótima escolha para o desenvolvimento do sistema proposto.

Conforme Furgeri (2008), o sucesso de Java no mercado deve-se a algumas características referentes à orientação a objetos, portabilidade, suporte à comunicação em rede e acesso remoto a banco de dados conforme é descrito:

i. A orientação a objetos permite o reuso de trechos de código e já é um paradigma sólido no mercado. Os objetos permitem a modelagem do mundo real;

ii. A portabilidade é uma conseqüência de a linguagem ser multiplataforma, ou seja, os programas são executados em qualquer dispositivo que contenha uma máquina virtual Java, sendo assim, as aplicações podem ser executadas em plataformas diferentes sem a necessidade de o código ser compilado novamente;

iii. O conceito de *Multithreading*, o qual permite a execução de um mesmo programa várias vezes ao mesmo tempo, e cada execução pode processar um trecho diferente do mesmo programa;

iv. Fornece suporte à comunicação em rede, encapsula detalhes de programação em classes prontas com funcionalidades específicas para suportar tecnologias avançadas de comunicação como os protocolos *Transmission Control Protocol/Internet Protocol* (TCP/IP), *Hyper Text Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP), entre outros;

v. Possui classes prontas para realizar acesso a banco de dados através de *drivers* que criam conexões com os bancos de dados.

O ambiente computacional Java possui um variado conjunto de tecnologias, conforme Jendrock *et. al.* (2014), dentre os quais podemos citar:

1. *Java Plataform, Standard Edition* (Java SE) – é a base da plataforma, inclui o ambiente de execução e bibliotecas comuns.
2. *Java Plataform, Enterprise Edition* (Java EE) – é a edição indicada para o desenvolvimento de aplicações corporativas e para internet.
3. *Java Plataform, Micro Edition* (Java ME) – indicada para desenvolvimento de aplicações para dispositivos móveis e embarcados.

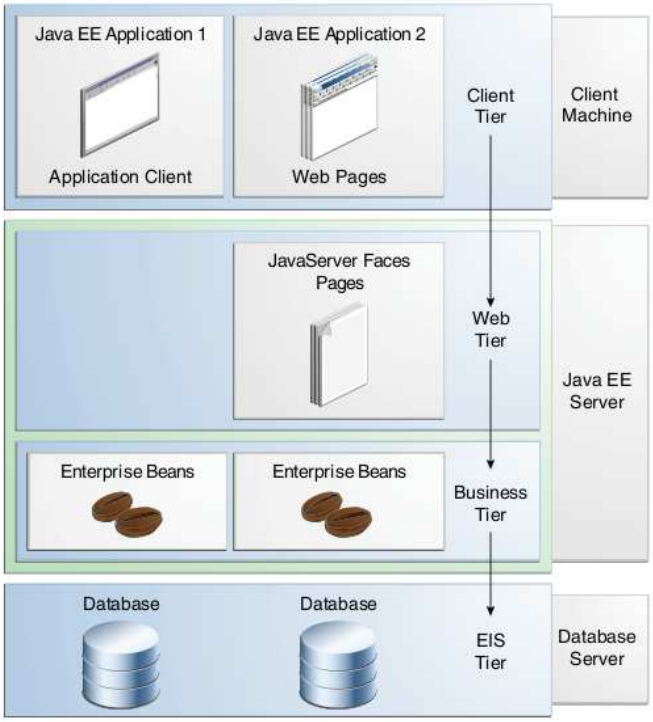
A próxima seção traz os conceitos da plataforma Java EE que foi escolhida para o desenvolvimento do presente estudo.

2.3 JAVA ENTERPRISE EDITION – JAVA EE

Na documentação da plataforma, Jendrock, Cervera-Navarro, Evans, Haase, Markito (2014) definem Java EE como um composto de uma máquina virtual Java e uma API que permite a execução de aplicativos escritos com o uso da linguagem de programação Java. Jendrock *et. al.* (2014) ainda cita que a arquitetura definida pela plataforma proporciona a implementação de serviços como aplicações multicamadas, sendo assim, a escalabilidade, a

acessibilidade e a capacidade de gerenciamento, necessários para aplicações de nível empresarial, são simplificadas, a figura 1 exemplifica aplicações multicamadas.

Figura 1 – Aplicações Multicamadas



Fonte: https://goo.gl/lqb9Km

O principal objetivo da plataforma é simplificar o desenvolvimento e melhorar a produtividade dos desenvolvedores, conforme Jendrock (2014). Para isso, ela inclui os seguintes recursos:

- Aplicações em lote (*batch aplication*);
- Utilitários de concorrência;
- API Java para processamento JSON;
- API Java para *WebSocket*;
- Recursos para componentes EJB;
- Recursos de *servlets* (Java Servlet Technology);
- Recursos para componentes JavaServer Faces;
- Recursos para a API Java Message Server (JMS).

Foi criada para aplicações empresariais complexas que mudavam frequentemente e necessitavam ser robustas. Surgiu em meados de 1999, num cenário de componentes distribuídos onde as aplicações teriam de se adaptar a novas soluções técnicas como serviços

web SOAP ou RESTful. Com o passar do tempo, tornou-se uma plataforma rica, destinada ao desenvolvimento de aplicações distribuídas, robustas, poderosas e altamente disponíveis (GONÇALVES, 2011).

A arquitetura Java EE é um conjunto de especificações implementadas que oferecem serviços aos componentes que ela hospeda, muito bem definida por Gonçalves:

> O Java EE é um conjunto de especificações implementadas por diferentes contentores. Contentores são ambientes de tempo de execução do Java EE que oferecem certos serviços aos componentes que eles hospedam, tais como gerenciamento de ciclo de vida, injeção de dependências, e assim por diante. Esses componentes usam contratos bem definidos para se comunicarem com a infraestrutura do Java EE e com os outros componentes. (2011, p. 4)

A seção seguinte aborda os EJB's que possuem propriedades interessantes para abstrair a lógica funcional e proporcionar maior segurança à aplicação.

2.4 EJB

Na obra de Gonçalves (2007), a definição encontrada para EJB é que são componentes do lado do servidor que rodam um *container*[1] para EJB do servidor de aplicação. Nesse sentido, são considerados componentes principais da plataforma Java EE cujo objetivo é permitir o desenvolvimento de aplicações Java baseadas em componentes, distribuídas, transacionais, seguras e portáveis de modo simplificado e rápido.

Gonçalves (2011) elucida o conceito de EJBs sendo componentes do lado do servidor cuja lógica funcional é encapsulada com o princípio de tratarem segurança, transações e uma série de outros serviços. Com os EJBs, constrói-se uma camada funcional entre a camada de persistência e a camada de apresentação.

Jendrock *et.al.* (2014) definiram EJB como componentes Java EE do lado do servidor que encapsulam a lógica de negócios de um aplicativo. Por lógica de negócio, entende-se os trechos de código que cumprem o propósito do aplicativo. Os benefícios de utilizar *beans* corporativos para simplificar o desenvolvimento de aplicações grandes está no fato de o *container* EJB fornecer serviços no nível de sistema para eles, por exemplo, gerenciamento de

[1] Container: é um delimitador abstrato, ou seja, um objeto que contem outros objetos que podem ser incluídos ou removidos dinamicamente.

transações e autorizações de serviços, além de permitir ao desenvolvedor preocupar-se apenas com os problemas do negócio.

A próxima seção apresenta a JPA e sua relação com o mapeamento objeto-relacional.

2.5 JPA

Gonçalves (2011), em sua obra, afirma que a tecnologia JPA foi incluída no Java EE, e é uma API que resolve problemas de persistência de dados reunindo os modelos de orientação por objetos e relacional. Torna possível a independência do SQL, ao passo que é uma abstração sobre o JDBC. O autor elenca como principais componentes da API:

- O mapeamento de objetos para dados como mecanismo para armazenar dados em uma base de dados relacional;
- Gerenciamento de entidades para realizar operações de criar, ler, atualizar e excluir no banco de dados;
- A linguagem JPQL que realiza recuperação de dados com consultas orientadas por objetos;
- Controle de acesso concorrente com a API JTA através de mecanismos de transações e bloqueio;
- Inclusão de lógica funcional no ciclo de vida de um objeto persistente através de callbacks e escutadores.

Gomes define JPA de maneira simples e objetiva como sendo criada especificamente para tornar possível a persistência objeto-relacional com banco de dados relacionais:

> O JPA é a mais recente especificação criada para permitir persistência objeto-relacional com banco de dados relacionais. Ao invés de salvar dados em tabelas, o código do sistema pede a persistência de classes carregadas com os valores que se quer salvar. Assim, os produtos que implementam JPA, como o Hibernate, transformam as requisições de salvar ou consultar via classes, em comandos SQL que são enviados ao banco de dados. (2008, p. 11)

Jendrock *et. al.* (2014), cita que a API Java Persistence proporciona ao desenvolvedor uma facilidade de mapeamento objeto/relacional para gerenciar dados relacionais em aplicativos Java. Ainda, mostra que esta API consiste de quatro áreas:

- A Java Persistence API;
- A linguagem de consultas;

- A Java Persistence Criteria API;
- Metadados para mapeamento objeto/relacional.

Em sua obra, Gonçalves (2011), ainda, afirma que JPA é apenas uma especificação, sendo assim, necessita de uma implementação. Dentre as soluções de mapeamento objeto-relacional mais famosas podemos citar: o TopLink, a Persistência Gerenciada de Contentor de Entidades (CMP), o Hibernate e o eclipse link.

O servidor de persistência Hibernate é o tema da próxima seção, bem como sua relação com a JPA.

2.6 HIBERNATE

Para Bauer e King (2007), um sistema de informação tem fundamento prático, se preservar os dados após ser desligado. Normalmente, os dados são guardados em bancos de dados com o uso de SQL tornando, assim, os dados persistentes. Os diversos sistemas de gerenciamento de banco de dados, usados hoje, possuem interfaces distintas para uso de SQL, nesse sentido, o provedor de persistência Hibernate foi criado, a partir de uma implementação de código aberto, para mediar a interação entre a aplicação e o banco de dados. Ele é um dos provedores de persistência disponíveis para uso com a JPA.

Portanto, o Hibernate é audacioso, pois tem o objetivo de fazer o relacionamento chamado mapeamento objeto/relacional com o banco de dados, dessa forma, o desenvolvedor abstrai o modelo relacional e preocupa-se, apenas, com a lógica do negócio. É necessário seguir algumas regras para persistir suas classes, a fim de que uma mudança de banco de dados seja facilmente implementada alterando apenas algumas configurações do Hibernate. (Gonçalves, 2007).

Desenvolvedores Java de vários lugares do mundo, liderados por Gavin King, criaram o Hibernate. Mais tarde, a empresa JBoss Developer, comprada pela Red Hat, contratou alguns dos desenvolvedores para continuarem o projeto Hibernate. Hoje, em sua versão 4.3.10 apresenta melhor escalabilidade, mais confiança é altamente configurável e extensível e suporta a versão 2.1 da JPA. Na seção seguinte são apresentados os conceitos de JSF.

2.7 JAVASERVER FACES

Um *framework* de aplicativo web para simplificar o desenvolvimento da interface com o usuário e separar a apresentação da lógica de negócio, essa foi a definição dada por Deitel e Deitel (2010) para JavaServer Faces. Ainda, segundo os autores, adiciona-se componentes em uma página JSP através de duas bibliotecas de *tags* personalizadas JSP que são fornecidas pelo JSF. E, embora os componentes JSF padrão sejam suficientes para criar aplicativos, é possível escrever bibliotecas de componentes e utilizar projetos de código-fonte aberto e de fornecedores independentes.

Em sua obra, Gonçalves (2007) declara que a API JavaServer Faces foi projetada para facilitar o desenvolvimento de aplicações para web rapidamente, visto que o uso de componentes de interface de usuário (GUI) e a conexão desses componentes com objetos de negócio são simplificados, além disso o uso de JavaBeans e a navegação de páginas são facilmente automatizados.

Gonçalves explica a arquitetura do JavaServer Faces de maneira muito simples:

> A arquitetura do JSF é fácil de se entender, se você estiver familiarizado com estruturas web. Aplicações JSF são aplicações web padrão que interceptam HTTP através do servlet Faces e produzem HTML. Nos bastidores, a arquitetura permite que você se conecte com qualquer linguagem de declaração de páginas (PDL) ou linguagem de declaração de vistas (VDL), execute-a para diferentes dispositivos (navegador web, dispositivos portáteis, etc.), e crie páginas usando eventos, escutadores e componentes à La Swing. (2011, p. 308)

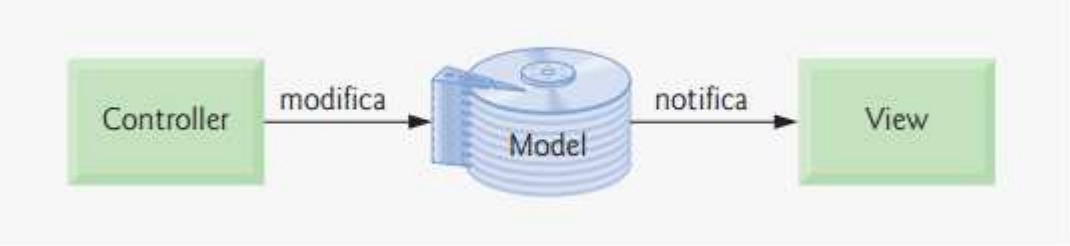
Alguns componentes do JSF são simples, como campos de entrada de dados e botões, enquanto outros são bastante sofisticados, como, por exemplo, tabelas de dados e árvores. O padrão MVC é descrito na próxima seção.

2.8 O PADRÃO MODEL-VIEW-CONTROLLER - MVC

A liberdade para pensar partes específicas dos sistemas foi alcançada através dos padrões de projeto que permitiram aos desenvolvedores projetarem seus modelos de forma mais abstrata, ou seja, promoveram o fraco acoplamento entre os objetos (DEITEL; DEITEL, 2010). Segundo os autores, no modelo arquitetônico MVC o modelo é separado da apresentação, ao passo que os dados do aplicativo ficam separados dos componentes gráficos. Da mesma forma, o controlador separa a lógica de processamento. Sendo assim, os componentes ficam resguardados, pois os desenvolvedores conseguem modificar cada

componente individualmente. A figura 2 representa a relação dos componentes do padrão MVC.

Figura 2 – Padrão de Projeto MVC



Fonte: http://goo.gl/gqf5Ki

O padrão de projeto MVC, segundo Gonçalves (2011), isola a lógica funcional da interface do usuário e torna a aplicação mais fácil de manter e mais escalável. Aplicando-se o padrão arquitetural MVC, as manutenções tanto da aparência visual quanto das regras funcionais são facilitadas e a alteração de uma camada não afeta a outra. Enquanto, para Gamma, Helm, Jonson, Vlissides (2000), a fim de separar a Visão e os Modelos, um protocolo do tipo inserção/notificação (*subscribe/notify*) é estabelecido pelo padrão MVC.

Gamma *et al*. (2000), ainda defendem que antes do MVC os projetos de interface tendiam a agrupar todos os objetos em uma camada. Na definição apresentada em sua obra, os objetos persistentes da aplicação são o Modelo, a apresentação na tela é a Visão e o Controlador é o objeto que gerencia as interações entre a visão e o modelo.

A próxima seção descreve os sistemas de bancos de dados e o sistema de gerenciamento de banco de dados (SGBD) PostgreSQL.

2.9 SISTEMAS DE BANCOS DE DADOS

Em sua obra, Date (2003) conceitua sistemas de bancos de dados como um sistema de manutenção de registros digitais – uma coleção de dados, ou, ainda, uma mina de dados – cujo objetivo é guardar informações para que em um determinado momento os usuários desse sistema possam manipular essas informações da maneira que melhor lhes convém, seja consultando, atualizando dados, ou, até mesmo excluindo registros.

Quando utilizado por empresas, o sistema de banco de dados permite controle centralizado de seus dados, de modo que problemas como integridade, redundância e segurança tornam-se triviais para os administradores de bancos de dados.

O autor, ainda apresenta algumas vantagens do uso de sistemas de bancos de dados, quais sejam:

i. Densidade: elimina a necessidade de papel;
ii. Velocidade: um processador pode manipular milhares de informações por segundo, já os humanos são extremamente limitados para realizarem tarefas monótonas e erram com maior freqüência;
iii. Atualidade: dispõe de informações precisas e atuais a qualquer instante;
iv. Proteção: a perda de dados e o acesso não autorizado são facilmente gerenciados;
v. A redundância pode ser reduzida e a inconsistência pode ser evitada.

Quando um banco de dados é percebido pelo usuário como tabelas e os operadores à disposição do usuário geram tabelas novas a partir de antigas, o sistema de banco de dados é dito relacional (DATE, 2003).

2.9.1 PostgreSQL

Conforme Neto (2007), o PostgreSQL é um sistema gerenciador de banco de dados – SGBD – objeto relacional de código aberto, que segue os padrões SQL ANSI-92, 96 e 99, que se destaca por seu alto desempenho, fácil administração e utilização intuitiva em projetos.

Além de possuir versões para diversas plataformas, o PostgreSQL permite a utilização de linguagem SQL, *triggers* (gatilhos), e ferramentas de programação e construção pertinentes aos SGBD's encontrados no mercado (Oracle, InterBase, SQL Server, etc.); nele, ainda, é possível utilizar "*Embedded* SQL" com pré-compilação. Outra característica é possuir *drivers* ODBC e JDBC para interface com ambientes e linguagens de programação, tais como: DBE, Borland Delphi, Pearl, XML e Java, por exemplo (NETO, 2007).

3 METODOLOGIA

3.1 SISTEMAS DE GESTÃO MÉDICA

Abaixo são, brevemente, analisados alguns softwares de gestão médica que já estão no mercado.

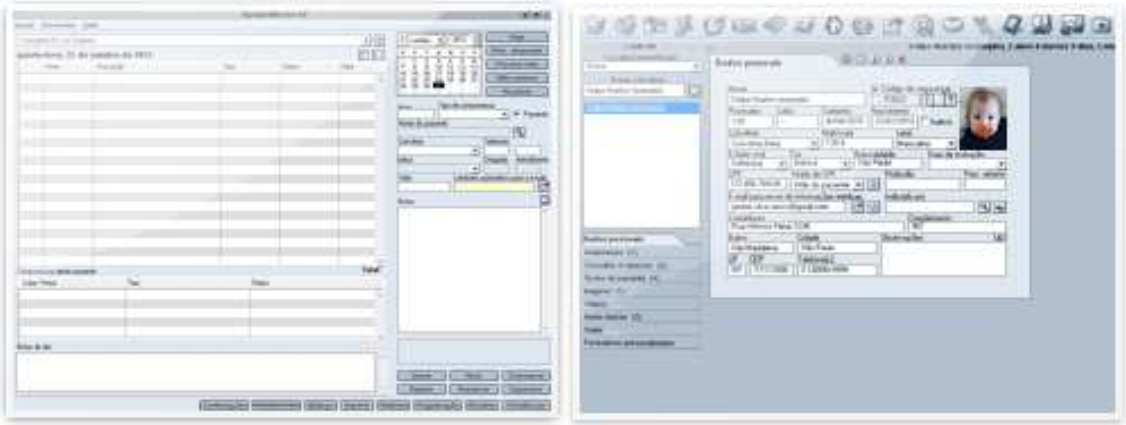
3.1.1 Hidoctor

O HiDoctor é um sistema de gestão médica do tipo cliente-servidor, desenvolvido pela empresa Centralx, que possui diversas funcionalidades, entre elas: agendamento, prontuário eletrônico completo, formulários personalizados, prescrição ágil e fácil, acesso a partir de estações Windows, pela web, iPhone e iPad; permitem sincronia de dados com a base *online*, confirmação de consultas, controle de tempo de paciente, atlas do corpo humano, faturamento por convênio, fila de atendimento de consultas, chat médico/secretária, site médico e estatísticas do consultório.

Conforme HiDoctor (2014), o software possui sincronização multiponto, com interfaces para celular e tablet, sendo assim, todos os computadores são atualizados.

Dois pacotes estão disponíveis para venda no site do fabricante, uma versão chamada HiDoctor e outra HiDoctor Office mais limitada que não inclui fila de adiantamento de consultas, relatórios de adiantamento de consultas, relatórios estatísticos de todos os agendamentos e relatórios estatísticos do site dos médicos, além de limitar em 5 páginas o site médico com área do paciente. A figura 3 apresenta as telas de agenda e de prontuário do sistema (HIDOCTOR, 2014).

Figura 3 – Telas de Agenda (esquerda) e de Prontuário (direita) do HiDoctor



Fonte: http://goo.gl/zC3Xcs

3.1.2 Medplus

A MedPlus é uma empresa brasileira de software, localizada no Paraná, especializada em sistemas de gerenciamento para clínicas médicas e soluções de atendimento para operadoras de planos de saúde. As soluções da empresa utilizam tecnologias em nuvem, ou

seja, as versões de software são totalmente web, permite gerenciamento multiclínicas e dentre as funcionalidades apontadas pelo fabricante estão: agenda médica interativa, confirmação de consulta via email, prontuário eletrônico, histórico clínico, controle financeiro de contas a pagar e a receber, armazenamento de imagens e de arquivos, sites personalizados, controle de estoque, emissão de nota fiscal de serviço eletrônica (NFS-e), relatórios gerenciais e estatísticos. A figura 4 mostra a tela de agendamento de consultas do sistema (MEDPLUS, 2014).

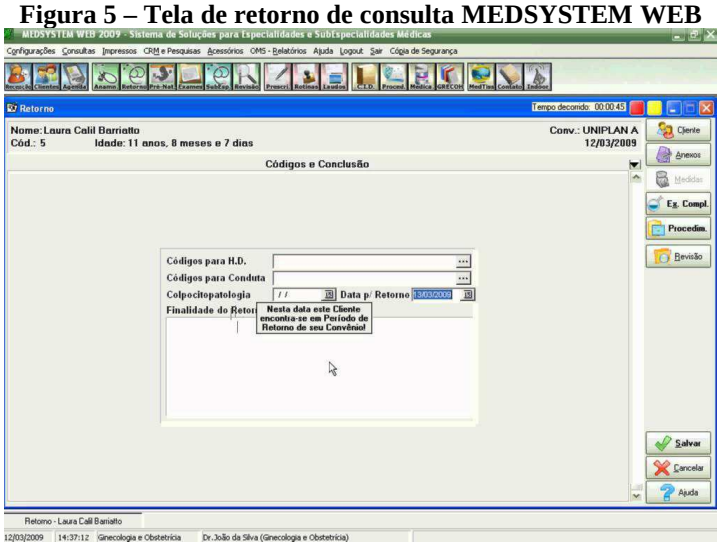
Figura 4 – Tela de Agendamento MedPlus

Fonte: http://site.medplus.com.br/

3.1.3 Medsystem Web

O sistema Medsystem Web é um software de gestão para consultórios médicos desenvolvido pela empresa brasileira MEDSYSTEM S/V de Minas Gerais, ele faz uso das metodologias RAD e SCRUM de desenvolvimento de software, e foi programado utilizando linguagens de programação C++, ASP, Framework IntraWeb, Java, além de utilizar os padrões do consórcio W3C de desenvolvimento para Web e sistema de gerenciamento de banco de dados Microsoft SQL Server. É um sistema do tipo cliente-servidor e tem como diferenciais: segurança, mobilidade e portabilidade. A figura 5 apresenta a tela de retorno de consultas do sistema (Medsystem Web, 2014).

Figura 5 – Tela de retorno de consulta MEDSYSTEM WEB



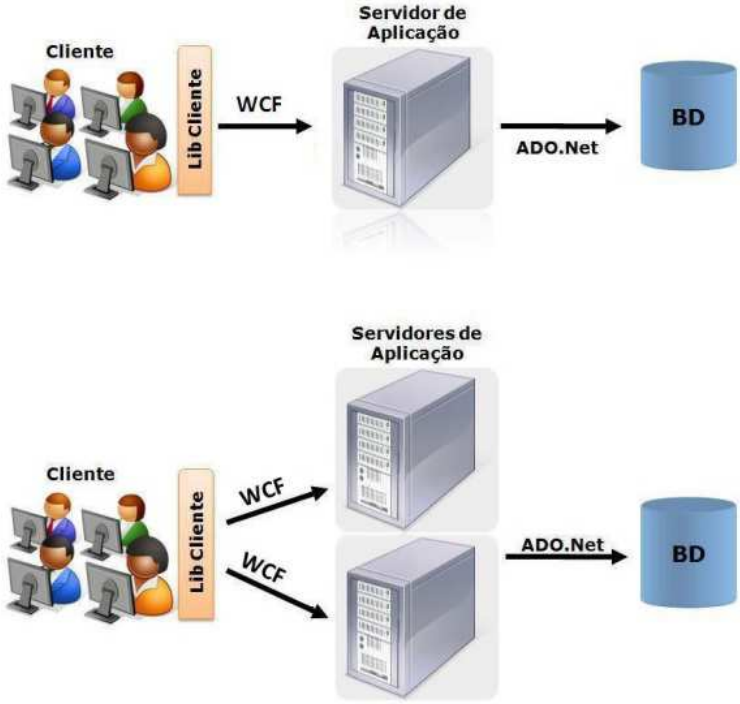
Fonte: https://goo.gl/vByWaT

3.1.4 Totvs Saúde

A TOTVS é uma empresa brasileira de software, líder no mercado brasileiro de ERP segundo a Fundação Getúlio Vargas, ela oferece produtos específicos e proprietários que atendem aos desafios dos mais variados segmentos. No ramo da saúde, a empresa propõe solução completa a qual integra todas as áreas de negócio e otimiza os processos. O software é desenvolvido, totalmente, utilizando linguagem .NET e, ainda, trabalha com a maioria dos bancos de dados no mercado, tais como, Oracle e MySQL.

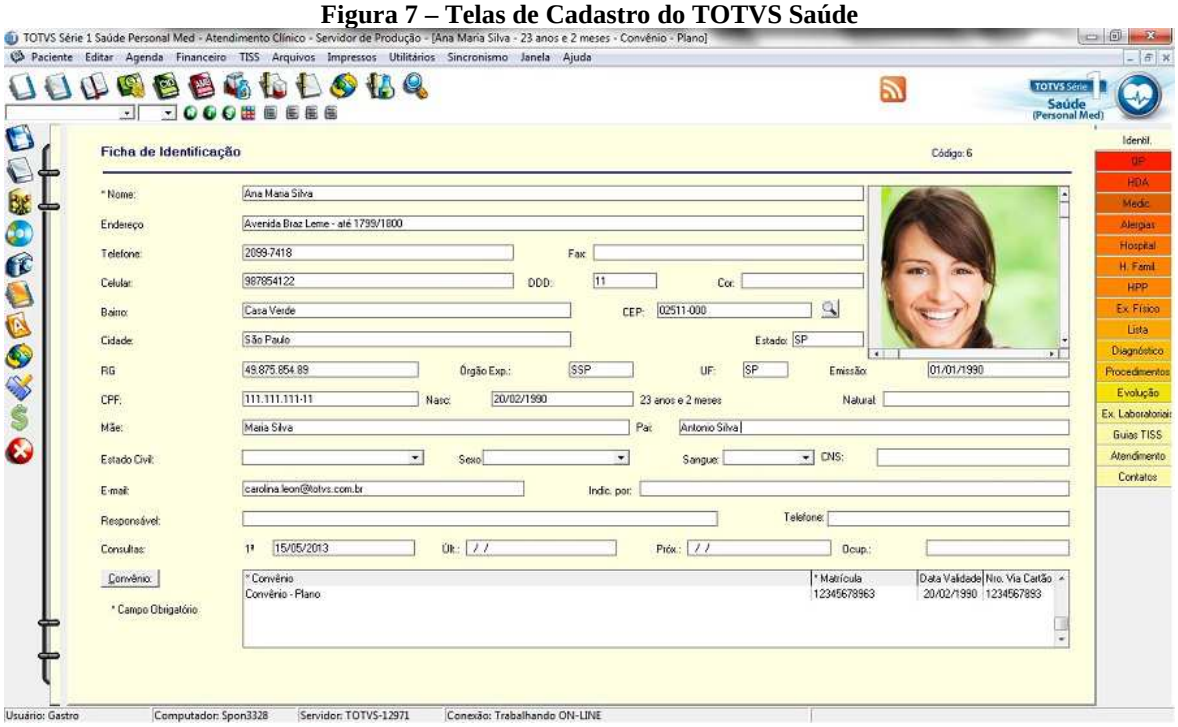
A arquitetura utilizada, em 3 (três) camadas conforme mostra a figura 6, diferencia-se da cliente-servidor, de forma que as tarefas são divididas possibilitando maior distribuição de processamento do sistema. A camada cliente dá acesso apenas aos formulários abertos na execução do sistema. Já a camada Servidor de Aplicação é responsável pela execução das regras de negócio do sistema (cálculos, processos, relatórios, além da comunicação com o banco de dados). Essa divisão em 3 (três) camadas, permite maior escalabilidade, reaproveitamento de regras de negócio, uso racional de hardware, balanceamento de carga, montagem de ambientes de reforço de contingência e diminuição da sobrecarga na camada cliente. Na figura 7, é mostrada a tela de cadastro do sistema (TOTVS, 2014).

Figura 6 – Arquitetura TOTVS em 3 camadas



Fonte: http://goo.gl/YnaEde

Figura 7 – Telas de Cadastro do TOTVS Saúde



Fonte: http://goo.gl/kODNhG

Da análise dos softwares citados, foi possível identificar que algumas funcionalidades são de extrema importância e não podem ficar fora de um sistema que almeja estar entre os mais vendidos no mercado. São elas: agendamento de consultas via web, prontuário

eletrônico, uma forma de confirmação de consultas e controle financeiro. A tabela 1 traz um comparativo entre os sistemas analisados e o sistema proposto.

Tabela 1 – Tabela comparativa dos sistemas analisados

	Hidoctor	**Medplus**	**Medsystem web**	**Totvs Saúde**	**Sistema Proposto**
Tecnologias	Não informado	Microsoft .NET com BD SQL Server	C#, .NET, Microsoft SQL Server	.NET	Java EE, BD PostgreSQL
Prontuário Eletrônico	Sim	Sim	Sim	Sim	Sim
Agendamento de Consultas On line	Não	Sim	Sim	Não	Sim
Confirmação de consulta	SMS	SMS	SMS, e-mail	Não	SMS, e-mail
Módulo Financeiro	Sim	Sim	Sim	Sim	Sim
Multiplataforma	Não	Sim	Sim	Sim	Sim
Licença	Compra do software e mensalidade	Pagamento de mensalidade	Pagamento de mensalidade	Pagamento de licença	Pagamento de mensalidade

3.2 DESENVOLVIMENTO

O desenvolvimento do sistema de gestão médica para web exigiu, inicialmente, o levantamento bibliográfico sobre a linguagem de programação, o paradigma de orientação à objetos e o banco de dados utilizados, neste caso, Java e PostgreSQL.

Com o objetivo do sistema web definido – um sistema de gestão que permita agendamento on line – realizou-se o levantamento de requisitos junto ao cliente através de entrevistas que conduziram ao desenvolvimento de modelo de casos de uso, posteriormente, fez-se a análise de requisitos, a modelagem do sistema e, por fim, o sistema.

3.3 REQUISITOS DO SISTEMA PROPOSTO

Uma clínica de fisioterapia necessita de um sistema de informação para dar apoio à realização de suas atividades principais, tais como: marcação de consultas, confirmação de consultas, armazenamento de histórico dos pacientes em meio eletrônico, e, ainda, controle de contas a pagar e a receber.

Além disso, a clínica quer fornecer o serviço de solicitação de agendamento de consultas pela internet e, assim, disponibilizar pesquisa a diversas informações dos fisioterapeutas que prestam serviço nessa casa de saúde. Também é propósito do sistema, publicar matérias informativas sobre métodos e patologias na página principal.

Essa clínica deseja um sistema de informação para gerenciar o atendimento aos seus pacientes, a fim de eliminar prontuários físicos e facilitar a consulta ao histórico dos pacientes. O negócio principal da clínica é a realização de consultas fisioterapêuticas. Para a realização da primeira consulta, o paciente deve informar seus dados – nome completo, endereço, complemento, bairro, CEP, email, sexo, registro geral, CPF, telefone, data de nascimento, apelido, senha, cidade, estado, profissão, nome do cônjuge, telefone do cônjuge, plano de saúde. O agendamento da primeira consulta deve ser por telefone ou pessoalmente, nas demais consultas, pode-se solicitar agendamento através da internet, conforme disponibilidade do fisioterapeuta desejado.

Na data da consulta, o fisioterapeuta tem acesso ao prontuário eletrônico do paciente e pode inserir dados relativos àquela consulta. Pode, ainda, prescrever tratamentos que farão parte do histórico do paciente. Além do histórico do paciente, o fisioterapeuta terá acesso há um histórico de medicações que o paciente utiliza, bem como as dosagens prescritas.

Os valores das consultas fisioterapêuticas serão definidos conforme a classe do paciente, que pode ser beneficiário de plano de saúde ou particular.

O paciente poderá pesquisar, via internet, o nome dos fisioterapeutas que atendem na clínica, bem como suas especialidades, e os horários disponíveis nas suas agendas. O paciente só pode solicitar marcação de consulta em um horário disponível na agenda do fisioterapeuta. As consultas marcadas podem ser canceladas pelo paciente em até 24h, caso não compareça e não cancele, terá que pagar meia consulta antes de agendar uma nova. Os valores das consultas estarão disponíveis para consulta no sistema web.

O sistema, também deve permitir a impressão de relatórios de consultas agendadas por semana, por mês, por fisioterapeuta, relatório de atendimentos por especialidade em um mês,

relatório de contas a receber, relatório de contas a pagar, valor total de consultas de um determinado paciente.

3.4 LEVANTAMENTO DE REQUISITOS

O levantamento de requisitos, descrito por Bezerra (2007) como um estudo exploratório das necessidades dos usuários e da situação do sistema atual, foi realizado através de observação do ambiente do usuário, realização de entrevistas com o usuário, entrevistas com especialistas do domínio, além de comparação com sistema preexistente do mesmo domínio do negócio.

3.4.1 Requisitos Funcionais

RF1. O sistema deve registrar consultas, indicando o paciente e o fisioterapeuta, bem como a data, a hora, o valor da consulta e o plano de saúde do paciente.
RF2. O sistema deve permitir que um fisioterapeuta insira informações no prontuário de um paciente, e essa ação deve ser registrada com a data e hora da inserção.
RF3. O sistema deve permitir que os pacientes cancelem suas consultas em até 24h de antecedência sem custo algum, no entanto, quando este prazo não for respeitado ou o paciente não comparecer à consulta, o sistema deve gerar uma fatura no valor de meia consulta e bloquear o agendamento para este paciente até que ele quite sua dívida.
RF4. Os fisioterapeutas devem ter acesso a relatórios de consultas agendadas, bem como ao número de consultas realizadas em determinado período.
RF5. O sistema deve permitir que o fisioterapeuta dê descontos e permita pagamento a prazo em função da política da clínica de fisioterapia.
RF6. O sistema deve permitir que um funcionário do consultório registre o pagamento das consultas, agende e cancele consultas, realize cadastros de pacientes, fisioterapeutas, medicamentos, profissões e planos de saúde.
RF7. Somente um fisioterapeuta pode desbloquear o agendamento de consultas de pacientes inabilitados de realizar marcação.
RF8. O sistema deve permitir consultas às agendas dos fisioterapeutas e aos planos de saúde aos quais esses são conveniados. Essas consultas poderão ser feitas informando uma ou mais informações, dentre elas: o nome do fisioterapeuta, o plano de saúde, a data.

RF9. O sistema deve manter histórico das consultas e, portanto, pacientes que tenham realizado alguma consulta não poderão ser excluídos.
RF10. O sistema deve permitir ativar e desativar pacientes e fisioterapeutas. Ainda, deve gerar um código de identificação único para cada paciente e fisioterapeuta.

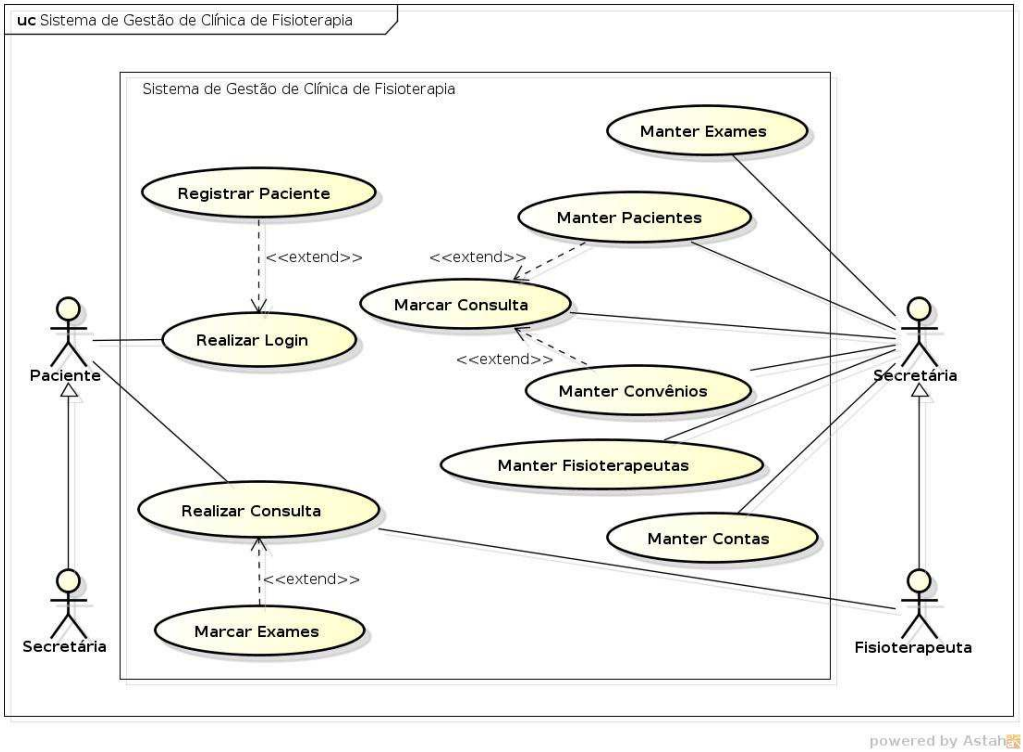
3.4.2 Requisitos Não Funcionais

RNF1. O sistema deve ser confiável, nesse sentido, deve atender as suas especificações com sucesso.
RNF2. A probabilidade de o sistema estar operacional num instante de tempo determinado deve ser alta.
RNF2. O software deve ser portável, ou seja, deve ser utilizado em qualquer plataforma de hardware e de software.
RNF3. O sistema deve tratar acessos não autorizados, sendo assim, deve garantir alto grau de segurança e, ainda, controlar o acesso às funcionalidades através de grupos de administradores e de usuários comuns.
RNF4. A interface do software deve ser amigável, ou seja, o usuário deve se sentir confortável ao utilizar o sistema, de forma que sua experiência torne-se fácil.
RNF5. A consulta à agenda deve estar disponível pela Internet, a partir dos principais navegadores disponíveis no mercado.
RNF6. A persistência das informações deve ser implementada em um Sistema Gerenciador de Bancos de Dados Relacional (SGBDR) livre (PostgreSQL).

3.5 PROJETO DO SISTEMA

Do levantamento e da análise de requisitos realizados, chegou-se ao diagrama de casos de uso da figura 8, que conforme GUEDES (2011) apresenta uma visão externa geral das funcionalidades que o sistema deverá oferecer aos usuários, sem se preocupar com a questão de como tais funcionalidades serão desenvolvidas.

Figura 8 – Diagrama de Casos de Uso



Fonte: Elaborada pelo Autor

O paciente pode realizar *login* no sistema e realizar uma consulta. A secretária além de poder manter os exames, os pacientes, os fisioterapeutas, os convênio e as contas, pode marcar consultas, além de, por ser um ator filho do paciente, herdar todos os casos de uso dele. O fisioterapeuta é um ator filho da secretária, logo, herda todos os casos de uso dela.

Nas tabelas 1, 2, 3 e 4 serão documentados os 4 (quatro) casos de uso julgados fundamentais para o entendimento do sistema, quais sejam: Realizar *Login*, Marcar Consulta, Manter Paciente e Realizar Consulta.

Tabela 2 – Documentação do Caso de Uso Realizar *Login*

Nome do Caso de Uso	**Realizar *Login***
Caso de Uso Geral	
Ator Principal	Paciente
Resumo	Este caso de uso descreve as etapas percorridas por um paciente para realizar *login* no sistema.
Pré-Condições	O paciente deve estar cadastrado no sistema e possuir nome de usuário e senha.

Pós-Condições	O *login* é efetuado com sucesso.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. O paciente seleciona o menu *login*, no qual insere seu nome de usuário e sua senha.	
	2. O sistema valida o *login*. A1. A2.
3. Inicia sessão.	
Fluxo Alternativo A1	
	3. Usuário não está cadastrado no sistema.
	4. Retorna mensagem de erro.
Fluxo Alternativo A2	
	3. Usuário digitou senha errada.
	4. Retorna mensagem de erro.

Tabela 3 – Documentação do Caso de Uso Marcar Consulta

Nome do Caso de Uso	**Marcar Consulta**
Caso de Uso Geral	
Ator Principal	Funcionário
Ator Secundário	Fisioterapeuta
Resumo	Este caso de uso descreve as etapas percorridas por um funcionário para realizar a marcação de uma consulta.
Pré-Condições	O funcionário deve estar logado no sistema.
Pós-Condições	A Consulta é agendada com sucesso.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. O funcionário seleciona o menu de marcação de consulta. A1.	
	2. O sistema carrega a agenda dos fisioterapeutas com os horários

	disponíveis.
3. Seleciona o fisioterapeuta. A2.	
4. Seleciona o Paciente. A3.	
5. Seleciona o convênio. A4.	
6. Seleciona data e hora desejada. A5.	
7. Clica no botão salvar.	
	8. Apresenta as informações da consulta, tais como, valor, data e horário.
	9. Registra consulta.
Fluxo Alternativo A1	
Ações do Ator	**Ações do Sistema**
1. O funcionário seleciona a opção de marcação de consulta sem estar logado no sistema.	
	2. Apresenta mensagem de erro informando que ele deve realizar *login* no sistema.
Fluxo Alternativo A2	
3. Seleciona o paciente sem selecionar o fisioterapeuta.	
	4. Apresenta mensagem de erro informando que fisioterapeuta é campo obrigatório e deve ser informado.
Fluxo Alternativo A3	
4. Seleciona o convênio sem selecionar o paciente.	
	5. Apresenta mensagem de erro informando que paciente é campo obrigatório e deve ser informado.
Fluxo Alternativo A4	
5. Seleciona a data e hora sem selecionar o convênio.	

	6. Apresenta mensagem de erro informando que convênio é campo obrigatório e deve ser informado.
Fluxo Alternativo A5	
6. Clica no botão salvar sem inserir data e hora.	
	7. Apresenta mensagem de erro informando que data e hora são campos obrigatórios e devem ser informados.

Tabela 4 – Documentação do Caso de Uso Manter Paciente

Nome do Caso de Uso	**Manter Paciente**
Caso de Uso Geral	
Ator Principal	Paciente
Ator Secundário	Funcionário, Fisioterapeuta
Resumo	Este caso de uso descreve as etapas percorridas por um paciente para realizar atualização de dados no sistema.
Pré-Condições	O paciente deve estar logado no sistema.
Pós-Condições	O cadastro do paciente é atualizado com sucesso.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. O paciente seleciona a opção de atualização de dados no menu. A1. A2.	
	2. Carrega dados do paciente.
3. O paciente informa os dados a serem atualizados. A3. A4.	
	4. Apresenta detalhes do paciente.
5. Confirma alterações.	
	6. Registra atualização do paciente.
Fluxo Alternativo A1	

1. O paciente seleciona a opção de atualização de dados sem estar logado no sistema.	
	2. Apresenta mensagem de erro informando que o usuário deve estar logado no sistema.
Fluxo Alternativo A2	
	3. Apresenta mensagem de erro informando que o paciente não está registrado no banco de dados.
Fluxo Alternativo A3	
	3. Campo obrigatório não foi preenchido (nome, endereço, etc) informa que o campo obrigatório deve ser preenchido.
Fluxo Alternativo A4	
	3. Campo CPF preenchido com valor incorreto, informa através de mensagem.

Tabela 5 – Documentação do Caso de Uso Realizar Consulta

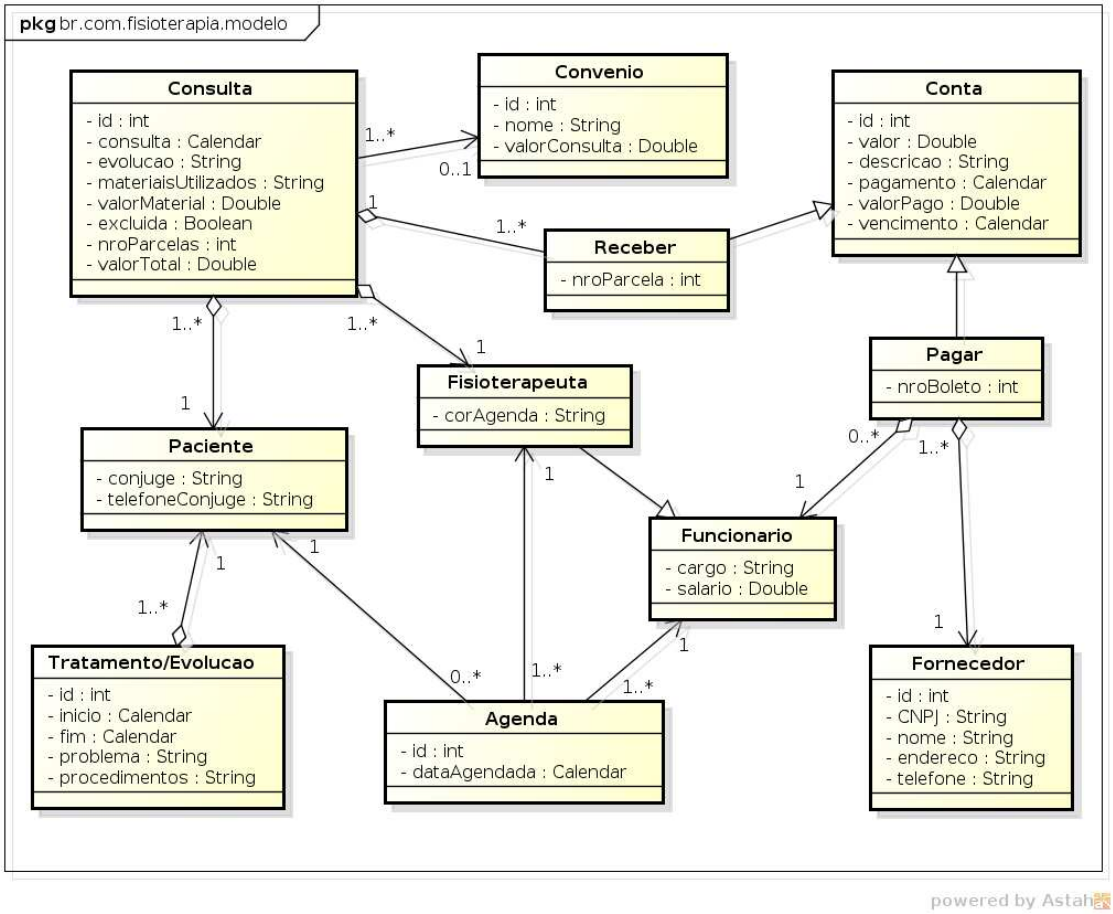
Nome do Caso de Uso	**Realizar Consulta**
Caso de Uso Geral	
Ator Principal	Fisioterapeuta
Ator Secundário	Paciente
Resumo	Este caso de uso descreve as etapas percorridas por um paciente para realizar consulta
Pré-Condições	O fisioterapeuta deve estar logado no sistema
Pós-Condições	O fisioterapeuta insere os dados do paciente na consulta agendada e essas informações são armazenadas no banco de dados.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. O fisioterapeuta seleciona a opção consulta. A1.	

	2. Apresenta a agenda do fisioterapeuta. A2.
3. Seleciona a consulta a ser realizada.	
	4. Apresenta detalhes do paciente.
5. Insere informações da consulta no campo evolução.	
	6. Solicita confirmação das informações inseridas.
7. Confirma alterações.	
	8. Registra alterações do paciente.
Fluxo Alternativo A1	
	2. Informa que o fisioterapeuta não está logado no sistema.
Fluxo Alternativo A2	
	3. Não há consultas agendadas, apresenta tela de consulta permitindo evolução de um novo paciente.
4. Informa o paciente. A3	
	5. Volta ao item 4 do fluxo principal.
Fluxo Alternativo A3	
	5. O paciente não está cadastrado, informa mensagem de erro, vai para tela de cadastro de paciente.
	6. Volta para item 4 do fluxo principal.

3.6 MODELAGEM DO SISTEMA

A fim de visualizar as classes, os atributos e os métodos que compõe o sistema, bem como a forma como elas se relacionam, complementam e transmitem informações entre si, foi desenhado o diagrama de classes da figura 9. Esse diagrama, conforme Guedes (2011), mostra de maneira estática como as classes estão organizadas e define suas estruturas lógicas.

Figura 9 – Diagrama de Classes do Domínio

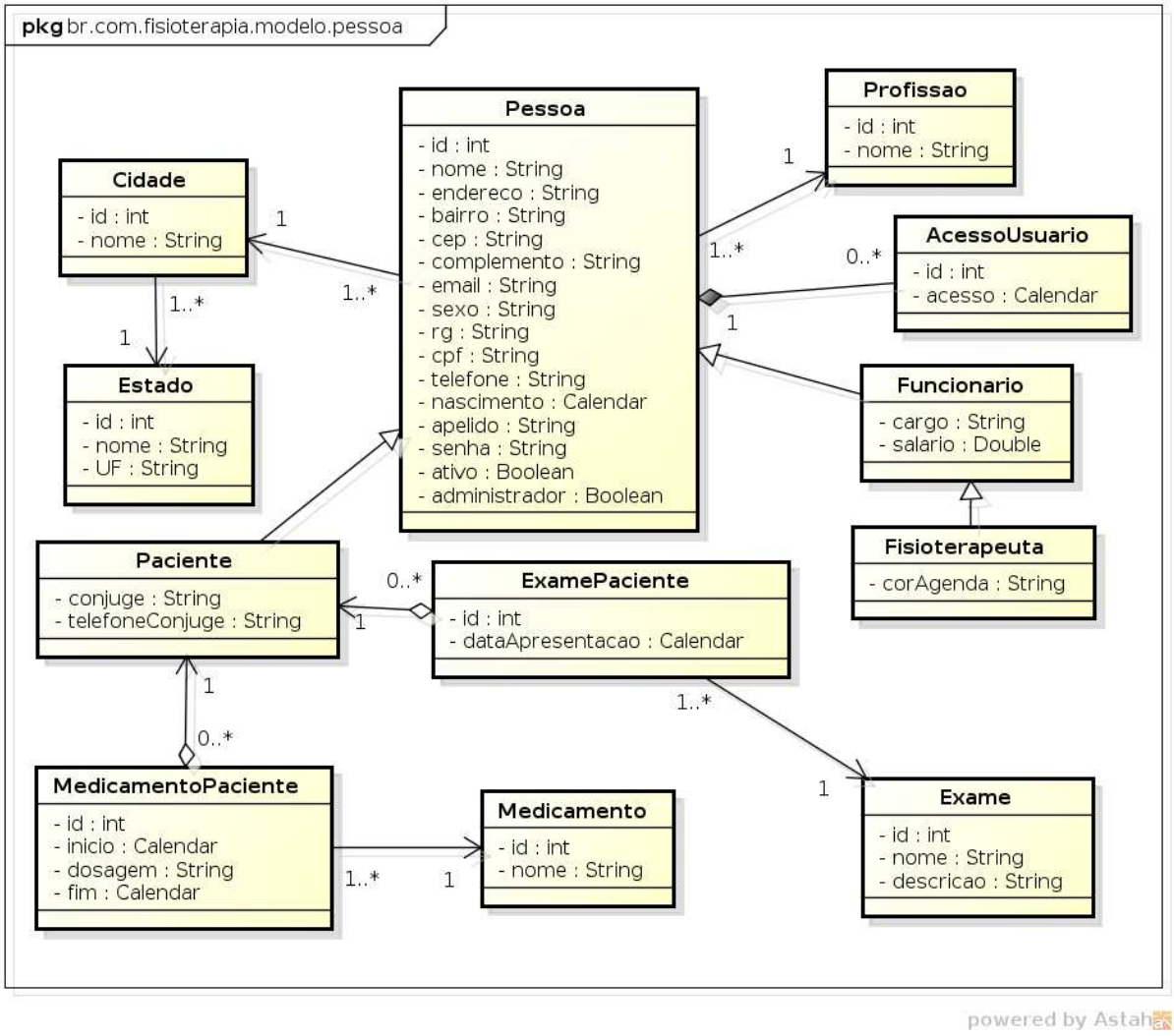


Fonte: Elaborada pelo Autor

A classe Agenda possui relacionamento com as classes Paciente, Fisioterapeuta e Funcionario, nessa relação uma agenda deve ter um paciente, um fisioterapeuta e um funcionário. A classe Tratamento/Evolução possui relacionamento com a classe Paciente no qual ela deve ter um paciente e a classe Paciente pode ter um ou muitos tratamentos. A classe Consulta possui relacionamento com as classes Convenio, Paciente e Fisioterapeuta, neles uma consulta deve ter um paciente e um fisioterapeuta e pode ter um ou nenhum convênio, em contrapartida, um convênio, um paciente e um fisioterapeuta podem ter uma ou muitas consultas. A classe Pagar possui relacionamento com a classe Funcionario, e, nesse caso, deve ter um funcionário e Funcionario pode ter uma ou muitas contas. A classe Conta é abstrata e possui duas classes filhas – Receber e Pagar – as quais possuem relacionamento com as classes Consulta e Fornecedor, nessas relações uma conta a receber deve ter uma consulta e uma conta a pagar deve ter um fornecedor, já as classes Consulta e Fornecedor podem ter uma ou muitas contas relacionadas.

Na figura 10, é apresentado o diagrama de classes do subsistema pessoa.

Figura 10 – Diagrama de Classes do subsistema pessoa



Fonte: Elaborada pelo Autor

A classe Pessoa é abstrata e possui duas filhas – Paciente e Funcionario – e se relaciona com as classes AcessoUsuario, Cidade e Profissao, nesses relacionamentos ela deve ter uma cidade e uma profissão, e pode ter nenhum ou muitos acessos. A classe Cidade se relaciona com Estado, assim, deve ter um estado e pode ter muitas pessoas na relação com a Pessoa. A classe Profissao pode ter muitas Pessoas e a classe AcessoUsuario deve ter uma pessoa.

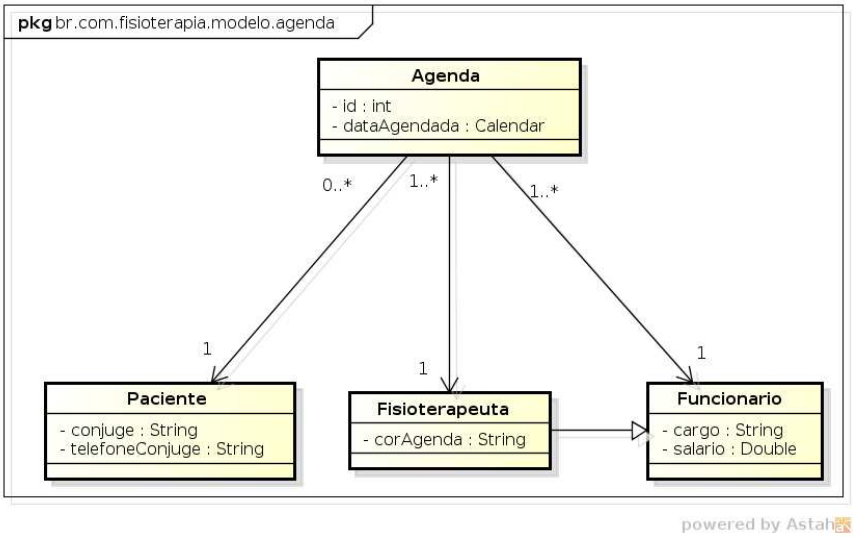
A classe Paciente, por ser filha de pessoa, herda seus atributos e ainda, pode ter nenhum ou muitos exames e medicamentos, sendo assim, relaciona-se com as classes ExamePaciente e MedicamentoPaciente.

A classe Funcionario herda os atributos de Pessoa e possui como filha a classe Fisioterapeuta que herda todos os seus atributos.

A classe MedicamentoPaciente se relaciona com as classes Paciente e Medicamento, ela deve ter um medicamento e um paciente. A classe ExamePaciente, e, da mesma forma, relaciona-se com as classes Paciente e Exame, ela deve ter um paciente e um exame.

Nas figuras 11, 12, 13 e 14, estão representadas, de maneira menos abstrata, as relações entre as classes Agenda, Consulta, Pagar e Tratamento/Evolucao, com o subsistema pessoa.

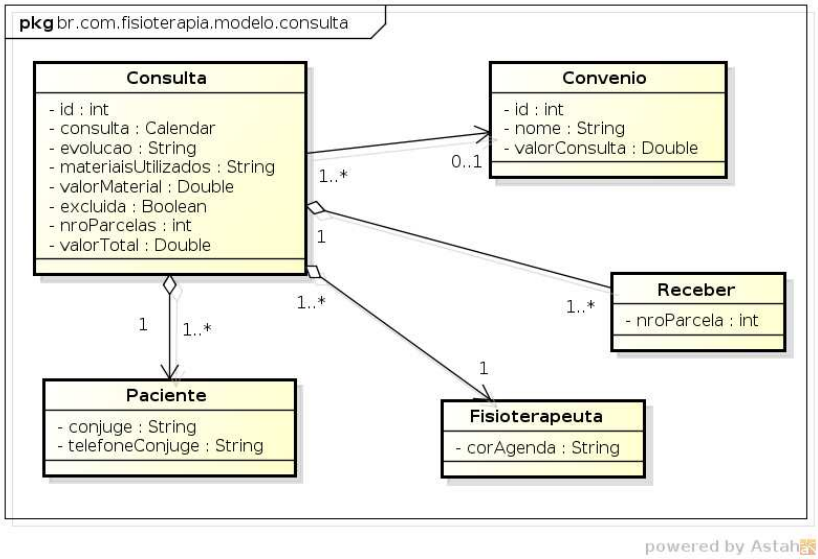
Figura 11 – Diagrama de Classes do Agendamento



Fonte: Elaborada pelo Autor

Nesse diagrama, a classe Agenda deve ter um fisioterapeuta, um funcionário e um paciente. Já a classe Paciente pode ter nenhuma ou muitas agendas, assim como, as classes Fisioterapeuta e Funcionario devem ter uma ou muitas agendas.

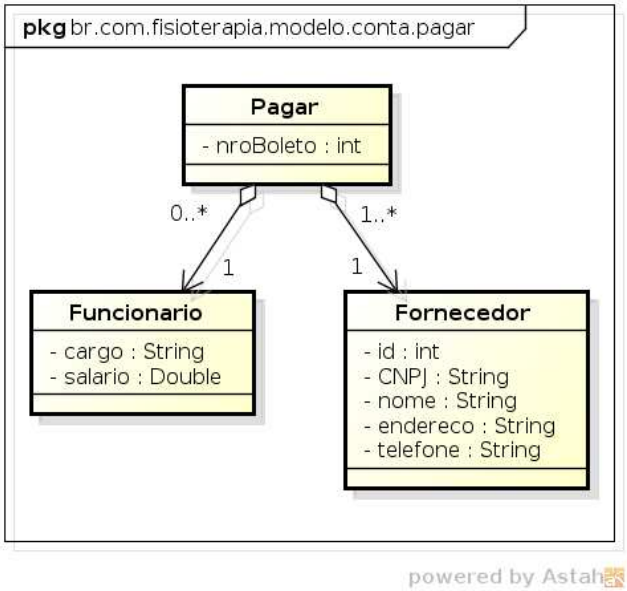
Figura 12 – Diagrama de Classes da Consulta



Fonte: Elaborada pelo Autor

Nessa figura, a classe Receber deve ter uma consulta e uma consulta pode ter uma ou muitas contas a receber. Já a classe Consulta pode ter nenhum ou um convênio, e deve ter um paciente e um fisioterapeuta, além disso, um convênio, um paciente e um fisioterapeuta podem ter uma ou muitas consultas.

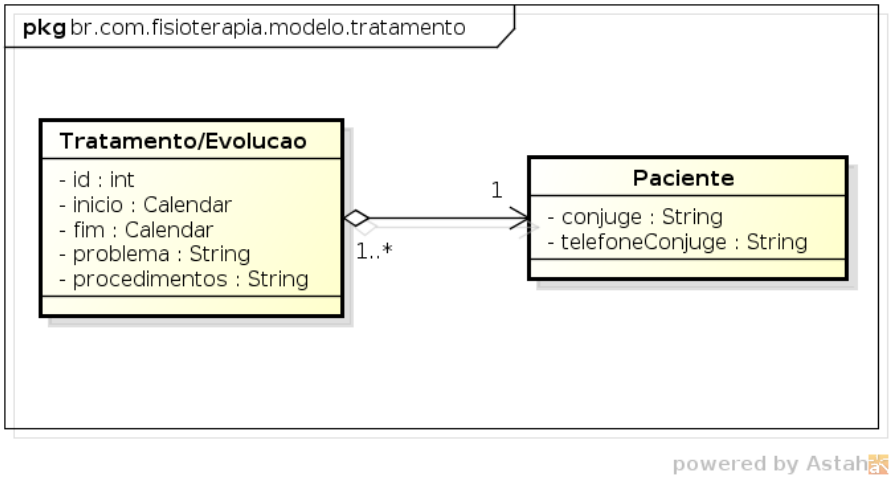
Figura 13 – Diagrama de Classes da Conta à Pagar



Fonte: Elaborada pelo Autor

Nesse diagrama, a classe Pagar deve ter um funcionario e um fornecedor, além disso, um fornecedor pode ter uma ou muitas contas a pagar e o funcionário pode ter nenhuma ou muitas contas a pagar.

Figura 14 – Diagrama de Classes do Tratamento



Fonte: Elaborada pelo Autor

Nesse relacionamento, a classe Tratamento/Evolucao deve ter um paciente e a Paciente, pode ter um ou muitos tratamentos.

3.7 PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

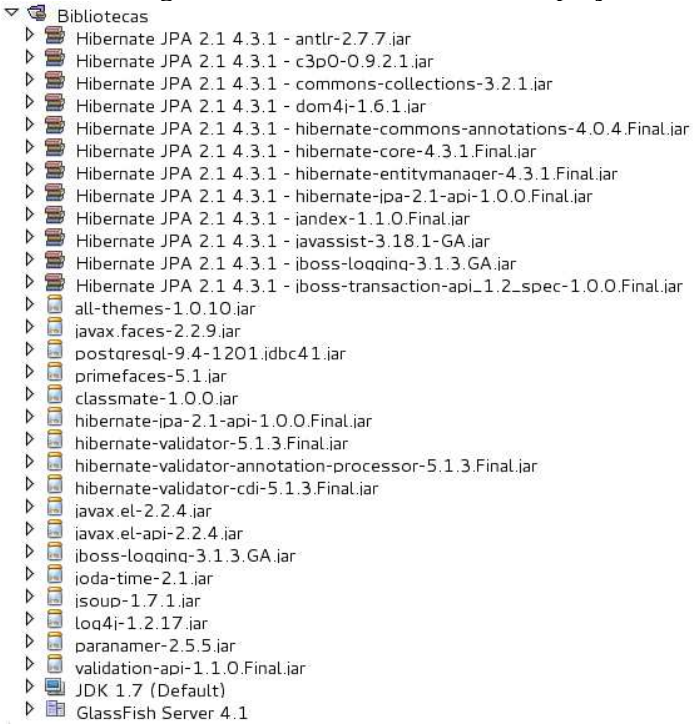
O sistema foi projetado seguindo o padrão de projetos MVC, portanto, nas sessões seguintes serão apresentadas a camada de Modelo, Visão e Controle. Para a programação do sistema, utilizou-se a versão 8.0.2 do NetBeans, o SGBD utilizado foi a versão 9.4 do PostgreSQL. O projeto criado no NetBeans é apresentado na figura 15.

Figura 15 – Projeto FisioterapiaERP no NetBeans

Fonte: Elaborada pelo Autor

As bibliotecas utilizadas foram a versão 4.3.1 do Hibernate que implementa a especificação 2.1 da JPA, a versão 5.1.2 do Hibernate Validator, a implementação Mojarra do JSF 2.2.9, a versão 5.1 do Primefaces e seus temas (all-themes-1.0.10) e a biblioteca do PostgreSQL. Todas elas são apresentadas na figura 16.

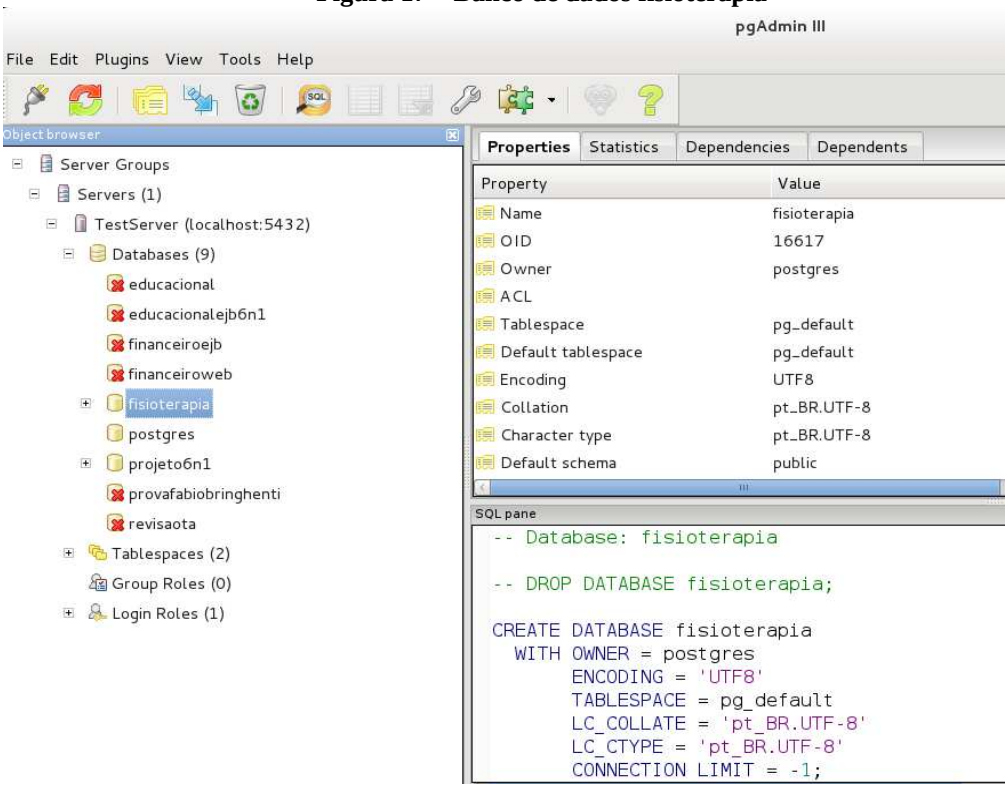
Figura 16 – Bibliotecas utilizadas no projeto



Fonte: Elaborada pelo Autor

O banco de dados foi criado através da ferramenta pgAdmin III e está apresentado na figura 17.

Figura 17 – Banco de dados fisioterapia



Fonte: Elaborada pelo Autor

Foi criado o arquivo de persistência, a fim de configurar a aplicação para utilizar o recurso JDBC do servidor GlassFish, e assim, delegar a ele o gerenciamento transacional dos objetos *Entity Manager*. A figura 18 mostra o código do arquivo *persistence.xml*, o qual é exigido pela especificação JPA, pois ele liga o provedor de JPA à base de dados.

Figura 18 – Código do arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.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="FisioterapiaERPPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/fisioterapiaerp</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.transaction.jta.platform"
                value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
    </properties>
  </persistence-unit>
  <persistence-unit>
</persistence>
```

Fonte: Elaborada pelo Autor

No elemento <*persistence-unit*> está definido o nome da persistência e o tipo de transação. Já o elemento <*provider*> define o provedor de persistência Hibernate, o elemento <*jta-data-sources*> define o nome da fonte de dados, a propriedade <*property name=”hibernate.hbm2ddl.auto” value=”update”*> indica que as tabelas serão atualizadas automaticamente quando o banco de dados for criado. A segunda propriedade define o dialeto do PostgreSQL e, assim, gera SQL otimizado para PostgreSQL. E a terceria propriedade define a plataforma de transação do Hibernate.

Já o arquivo *glassfish-resources.xml*, também criado com a unidade de persistência, define algumas propriedades do *pool* de conexões do servidor GlassFish e é apresentado na figura 19.

Figura 19 – Código do arquivo glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <jdbc-connection-pool allow-non-component-callers="false"
                          associate-with-thread="false"
                          connection-creation-retry-attempts="0"
                          connection-creation-retry-interval-in-seconds="10"
                          connection-leak-reclaim="false"
                          connection-leak-timeout-in-seconds="0"
                          connection-validation-method="auto-commit"
                          datasource-classname="org.postgresql.ds.PGSimpleDataSource"
                          fail-all-connections="false"
                          idle-timeout-in-seconds="300"
                          is-connection-validation-required="false"
                          is-isolation-level-guaranteed="true"
                          lazy-connection-association="false"
                          lazy-connection-enlistment="false"
                          match-connections="false"
                          max-connection-usage-count="0"
                          max-pool-size="32" max-wait-time-in-millis="60000"
                          name="fisioterapia_Pool" non-transactional-connections="false"
                          pool-resize-quantity="2" res-type="javax.sql.DataSource"
                          statement-timeout-in-seconds="-1" steady-pool-size="8"
                          validate-atmost-once-period-in-seconds="0" wrap-jdbc-objects="false">
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="5432"/>
        <property name="databaseName" value="fisioterapia"/>
        <property name="User" value="postgres"/>
        <property name="Password" value="CIC.2009"/>
        <property name="URL" value="jdbc:postgresql://localhost:5432/fisioterapia"/>
        <property name="driverClass" value="org.postgresql.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/fisioterapiaerp"
                   object-type="user" pool-name="fisioterapia_Pool"/>
</resources>
```

Fonte: Elaborada pelo Autor

No elemento <*jdbc-connection-pool*> o atributo *name* define o nome do recurso jdbc onde o servidor GlassFish irá instanciar as *Entity Managers*. As propriedades como nome do servidor, número da porta, nome da base de dados, usuário, senha, *url* e *driver* do banco de dados são definidas, também, nesse elemento.

Já o elemento <*jdbc-resource*> define em seu atributo *jndi-name* o nome do recurso que será utilizado pelos *beans* de sessão, o nome do *pool* de conexões no atributo *pool-name*.

3.8 MAPEAMENTO OBJETO-RELACIONAL

Conforme Gonçalves (2011, p. 53), a tarefa de criação de uma correspondência entre os objetos e as tabelas do sistema foi delegada à JPA que torna o mapeamento mais fácil com o uso de configuração por exceção. A figura 20 mostra a entidade Agenda com seus atributos e anotações necessários para ela ser persistida na base de dados, essa classe faz parte da camada de Modelo do padrão MVC.

Figura 20 – Entidade Agenda

```
 *
 * @author bringha
 */
@Entity
@Table(name = "agenda")
public class Agenda implements Serializable{
    @Id
    @SequenceGenerator(name = "seq_id_agenda", sequenceName = "seq_agenda_id", allocationSize = 1)
    @GeneratedValue(generator = "seq_id_agenda", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @NotNull(message = "A data e hora devem ser informadas")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "dataHora", nullable = false)
    private Calendar dataHora;
    @ManyToOne
    @JoinColumn(name = "funcionario", referencedColumnName = "id", nullable = false)
    private Funcionario funcionario;
    @NotNull(message = "O fisioterapeuta deve ser informado")
    @ManyToOne
    @JoinColumn(name = "fisioterapeuta", referencedColumnName = "id", nullable = false)
    private Fisioterapeuta fisioterapeuta;
    @NotNull(message = "O paciente deve ser informado")
    @ManyToOne
    @JoinColumn(name = "paciente", referencedColumnName = "id", nullable = false)
    private Paciente paciente;

    //construtores, getters e setters
}
```

Fonte: Elaborada pelo Autor

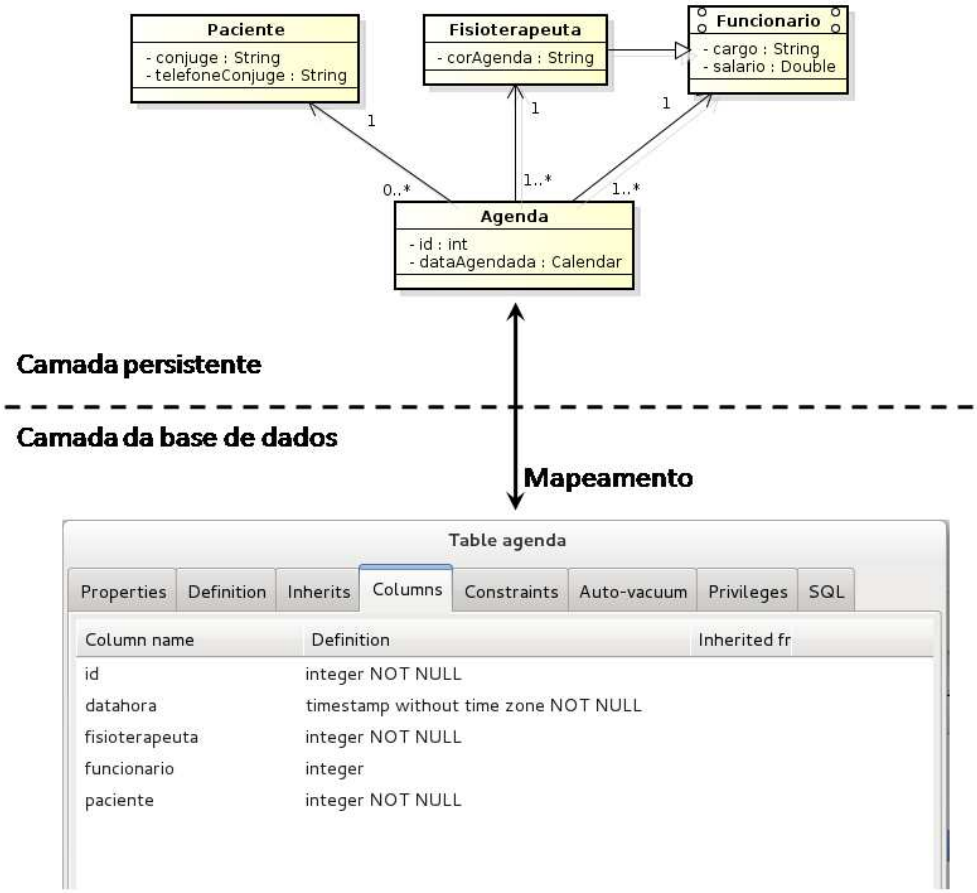
Para ser reconhecida como entidade, a classe Agenda deve ser anotada com *@Entity*. A anotação *@Id* é usada para denotar a chave primária, seu identificador é gerado automaticamente pelo provedor de persistência através da anotação *@GeneratedValue*, também, definiu-se a estratégia por *GenerationType.SEQUENCE*. Ainda, foi definido um nome para a sequência através da anotação *@SequenceGenerator*.

Para personalizar o mapeamento de uma coluna utilizou-se, por exemplo, a anotação *@Column(name = "dataHora", nullable = false)* que define o nome da coluna como dataHora e ela não pode ser nula. A anotação *@Temporal(TemporalType.TIMESTAMP)* indica que o atributo é do tipo *TIMESTAMP*. Já a anotação *@NotNull* pertence à API *Hibernate Validator* e indica uma restrição de integridade para que o atributo não seja nulo.

Para representar o relacionamento muitos para um entre a classe agenda e as classes Fisoterapeuta e Paciente, foram utilizadas as anotações *@ManyToOne* e *@JoinColumn(name = "paciente", referencedColumnName = "id", nullable = false)*; a primeira anotação define o relacionamento muitos para um e a segunda, o nome da coluna que será criada na tabela agenda, a referência que será utilizada, nesse caso a coluna id da tabela Paciente e a terceira indica a obrigatoriedade dos atributos.

Como mostra a figura 21, a entidade Agenda é armazenada em uma tabela Agenda, e cada coluna é nomeada de acordo com os atributos da classe de acordo com as anotações informadas no arquivo de modelo.

Figura 21 – Sincronização de dados entre a aplicação e o banco de dados



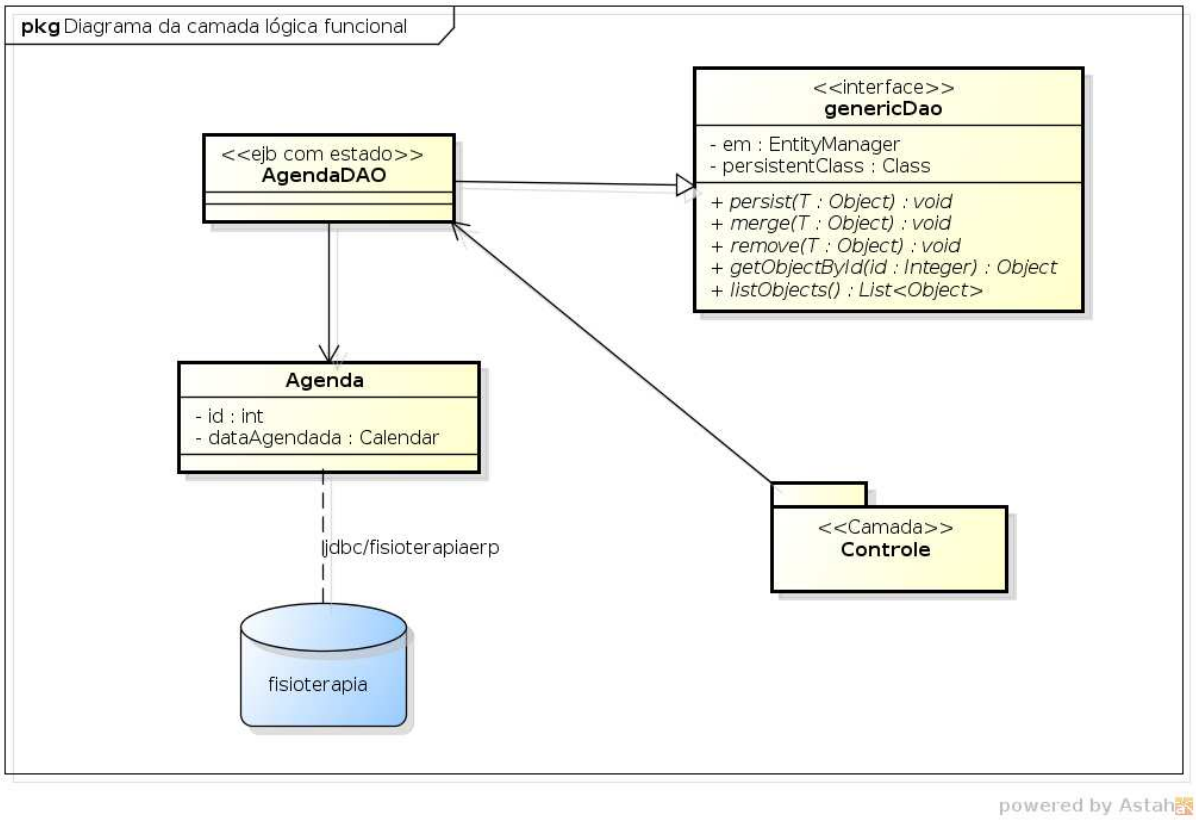
Fonte: Elaborada pelo Autor

3.9 A CAMADA LÓGICA FUNCIONAL

Com objetivo de criar uma camada funcional na aplicação, conforme orienta Gonçalves (2011), usou-se os EJBs que separam a camada de persistência da camada de apresentação, implementam a lógica funcional e adicionam gerenciamento de transações e segurança.

O diagrama da camada lógica funcional é apresentado na figura 22.

Figura 22 – Camada lógica funcional



Fonte: Elaborada pelo Autor

A classe AgendaDAO é um *bean* de sessão com estado, por natureza é transacional, e herda as operações de criação, leitura, atualização e exclusão (CRUD) da classe GenericDAO. As classes AgendaDAO e Agenda são empacotadas e distribuídas no servidor de aplicações GlassFish, assim, a aplicação, na sua camada de controle, invocará métodos no EJB.

Para utilizar as transações, o *bean* de sessão acessa a base de dados através da fonte de dados (jdbc/fisioterapiaerp), a qual foi criada no servidor GlassFish e ligada à base de dados fisioterapia.

A figura 23, mostra o código da classe GenericDAO, ela possui um atributo do tipo *Class* chamado *persistenceClass*, o qual guarda o tipo da classe que está utilizando a classe genérica; a anotação *@PersistenceContext(unitName = "FisioterapiaERPPU")* indica ao EJB o nome da persistência que será utilizada. O contentor também instancia um objeto *Entity Manager*, atributo chamado em, o qual é diretamente injetado no *bean* de sessão e, através dele, o contentor EJB realiza as transações necessárias. A injeção de dependências é um mecanismo simples, porém poderoso para injetar referências de recursos em atributos utilizado no Java EE 6.

Figura 23 – Código da classe GenericDAO

```
public class GenericDAO<T> implements Serializable {

    private Class persistentClass;
    @PersistenceContext(unitName = "FisioterapiaERPPU")
    private EntityManager em;

    public GenericDAO() {

    }

    public void persist(T object) throws Exception {
        em.persist(object);
    }

    public void merge(T objeto) throws Exception {
           em.merge(objeto);
    }

    public void remove(T objeto) throws Exception {
        objeto = em.merge(objeto);
        em.remove(objeto);
    }

    public T getObjectById(Integer id) throws Exception{
            return (T) em.find(persistentClass, id);
    }
    // getters, setter e demais métodos
}
```

Fonte: Elaborada pelo Autor

A figura 24 mostra o código da classe AgendaDAO, a qual possui a anotação *@Stateful* que define a classe como um *bean* de sessão com estado. O gerenciamento de instâncias é feito, automaticamente, pelo contentor EJB.

Figura 24 – Código da classe AgendaDAO

```
@Stateful
public class AgendaDAO<T> extends GenericDAO<T> implements Serializable{

    public AgendaDAO() {
        super();
        super.setPersistentClass(Agenda.class);
        //inicializar as ordenações possíveis
        super.getListOrder().add(
                new Order("id", "ID", "="));
        super.getListOrder().add(
                new Order("funcionario.nome", "Funcionario", "like"));
        super.getListOrder().add(
                new Order("fisioterapeuta.nome", "Fisioterapeuta", "like"));
        super.getListOrder().add(
                new Order("paciente.nome", "Paciente", "like"));
        // definir qual a ordenação padrão no caso o segundo elemento da lista (indice 1)
        super.setCurrentOrder((Order) super.getListOrder().get(1));
        // inicializando o filtro
        super.setFilter("");
        // inicializando o conversor da ordem
        super.setConverterOrder(new ConverterOrder(super.getListOrder()));
    }
}
```

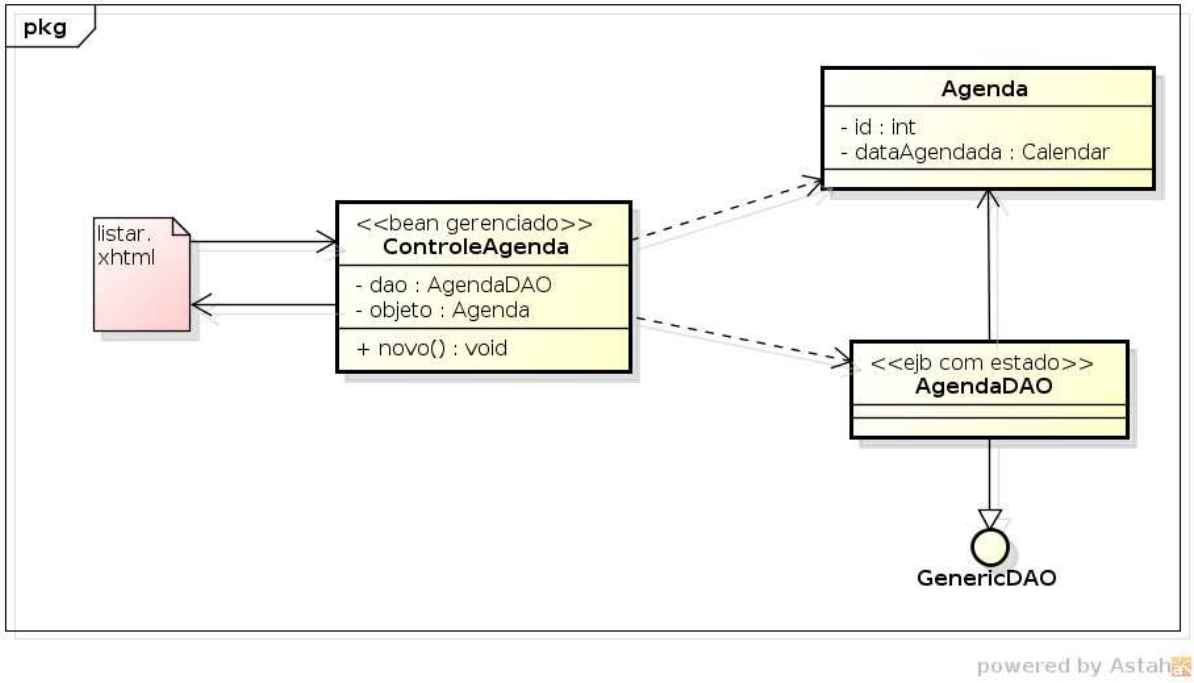
Fonte: Elaborada pelo Autor

O método *super.setPersistentClass(Agenda.class)* define o tipo de classe Agenda que está utilizando a classe GenericDAO. Os demais métodos são utilizados para realizar a paginação e ordenação dos registros.

3.10 ACOPLANDO A LÓGICA FUNCIONAL COM A PERSISTÊNCIA

Os *beans* gerenciados, segundo Gonçalves (2011), são usados para armazenar as propriedades necessárias para a navegação. Tudo pode ser conectado usando persistência com a JPA e lógica funcional com o EJB. A figura 25 mostra a interação dos componentes da aplicação, eles são empacotados em um arquivo *war*, distribuído em uma instância do GlassFish e uma base de dados ativa do PostgreSQL. Os *beans* gerenciados fazem parte da camada de Controle do padrão MVC.

Figura 25 – Páginas e classes envolvidas na aplicação



Fonte: Elaborada pelo Autor

O *bean* gerenciado delega toda a lógica funcional ao GenericDAO, que contém os métodos CRUD. Esse *bean* de sessão usa a API *Entity Manager* para manipular uma entidade Agenda, o ejb com estado herda todos os métodos e atributos de GenericDAO. A página listar.xhtml chama o método *listObjects( )* e recebe a lista de todas as agendas registradas na base de dados.

A função do *bean* gerenciado é interagir com as outras camadas da aplicação ou realizar validação. A figura 26 apresenta o código do *bean* gerenciado ControleAgenda, nele é possível identificar a anotação *@ManagedBean*, a qual indica que a classe é um *bean* gerenciado com o nome de controleAgenda.

Figura 26 – Código do bean gerenciado ControleAgenda

```
/**
 *
 * @author bringha
 */
@ManagedBean(name = "controleAgenda")
@ViewScoped
public class ControleAgenda implements Serializable{
    @EJB
    private AgendaDAO<Agenda> dao;
    private Agenda objeto;

    @EJB
    private FisioterapeutaDAO<Fisioterapeuta> daoFisioterapeuta;

    @EJB
    private FuncionarioDAO<Funcionario> daoFuncionario;

    @EJB
    private PacienteDAO<Paciente> daoPaciente;

    public ControleAgenda() {
    }

    //getters e setters

    public void novo(){...3 linhas }

    public void salvar(){...12 linhas }

    public void editar(Integer id){...7 linhas }

    public void remover(Integer id){...8 linhas }
}
```

Fonte: Elaborada pelo Autor

A segunda anotação *@ViewScoped* define a duração da vida do *bean* como escopo de visão, assim, ele dura mais que uma requisição e menos que uma sessão. Este *bean* contém quatro atributos dos tipos AgendaDAO, FisioterapeutaDAO, FuncionarioDAO e PacienteDAO injetados com referência ao EJB usando a anotação *@EJB* e tem um atributo objeto do tipo Agenda que será mapeado no formulário e persistido na base de dados.

Os métodos novo( ) e salvar( ) permitem a criação de uma agenda pela invocação do EJB. Os métodos editar( ) e remover( ) recebem um identificador por parâmetro e permitem a atualização e exclusão de um registro através da invocação do EJB.

Para esclarecer os métodos novo( ) e salvar( ) da classe ControleAgenda, a figura 27 apresenta seus códigos. Nela, é possível verificar que o método novo( ) instancia um objeto

agenda e o método salvar( ) verifica se objeto já existe na base de dados. Caso exista, o id do objeto não é nulo, o controlador invoca o objeto AgendaDAO, através de seu atributo dao com o método *merge*(objeto), assim, atualiza a base de dados. Se a agenda não existir, o id do objeto é nulo, o controlador invoca o objeto AgendaDAO, através de seu atributo dao com o método *persist*(objeto), assim persiste o novo objeto na base de dados.

Figura 27 – Código dos métodos novo e salvar

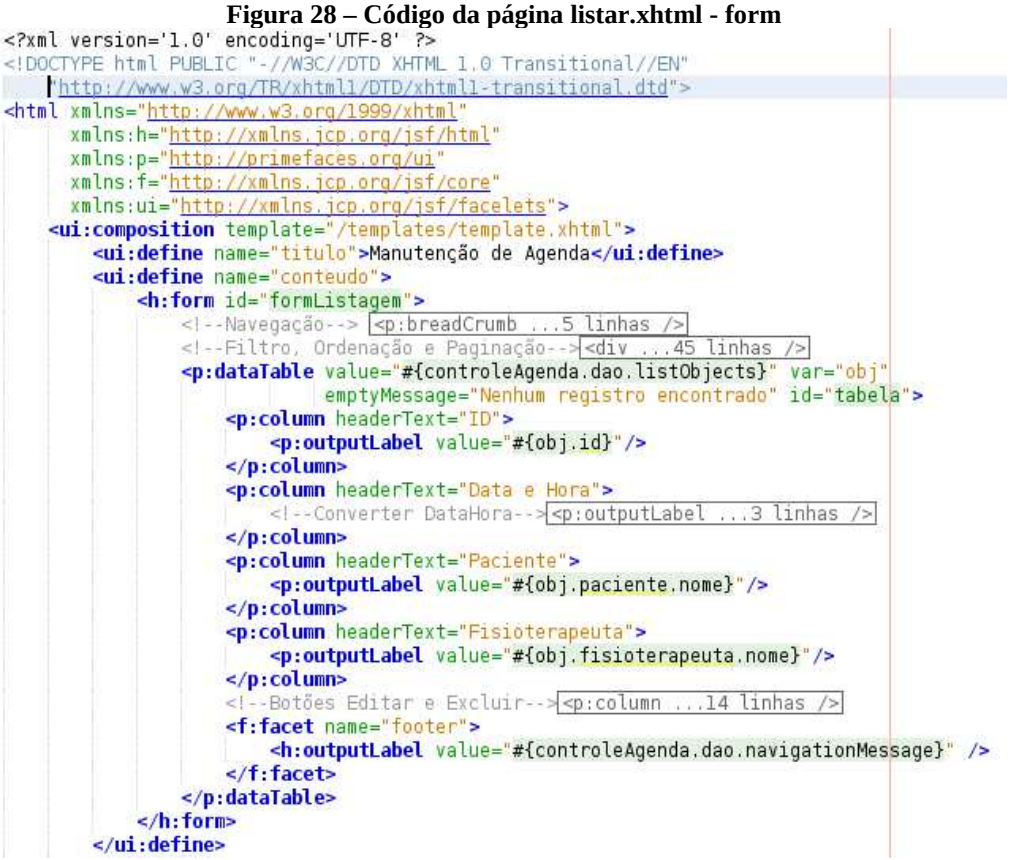
```
public void novo(){
    objeto = new Agenda();
}

public void salvar(){
    try{
        if(objeto.getId() == null){
            dao.persist(objeto);
        }else{
            dao.merge(objeto);
        }
        Util.mensagemInformacao("Objeto persistido com sucesso!");
    }catch(Exception e){
        Util.mensagemErro("Erro ao persistir o objeto: " + e.getMessage());
    }
}
```

Fonte: Elaborada pelo Autor

A página listar.xhtml, apresentada na figura 28, possui elementos da suíte de componentes *PrimeFaces*. Nela, inseriu-se um formulário que possui um elemento *dataTable* para receber os registros armazenados no banco de dados, além disso, um elemento *dialog* para que as inserções de novos registros sejam realizadas. A página listar.xhtml faz parte da camada de Visão do padrão MVC.

Figura 28 – Código da página listar.xhtml - form



```
<?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://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/templates/template.xhtml">
        <ui:define name="titulo">Manutenção de Agenda</ui:define>
        <ui:define name="conteudo">
            <h:form id="formListagem">
                <!--Navegação--> <p:breadCrumb ...5 linhas />
                <!--Filtro, Ordenação e Paginação--><div ...45 linhas />
                <p:dataTable value="#{controleAgenda.dao.listObjects}" var="obj"
                             emptyMessage="Nenhum registro encontrado" id="tabela">
                    <p:column headerText="ID">
                        <p:outputLabel value="#{obj.id}"/>
                    </p:column>
                    <p:column headerText="Data e Hora">
                        <!--Converter DataHora--><p:outputLabel ...3 linhas />
                    </p:column>
                    <p:column headerText="Paciente">
                        <p:outputLabel value="#{obj.paciente.nome}"/>
                    </p:column>
                    <p:column headerText="Fisioterapeuta">
                        <p:outputLabel value="#{obj.fisioterapeuta.nome}"/>
                    </p:column>
                    <!--Botões Editar e Excluir--><p:column ...14 linhas />
                    <f:facet name="footer">
                        <h:outputLabel value="#{controleAgenda.dao.navigationMessage}" />
                    </f:facet>
                </p:dataTable>
            </h:form>
        </ui:define>
```

Fonte: Elaborada pelo Autor

O elemento *<p:dataTable>* liga o atributo dao, método *listObjects*( ), do *bean* gerenciado (um *ArrayList* de agendas) e declara a variável obj para iterar a lista. Os atributos da agenda são apresentados nas colunas através das expressões #{obj.id}, #{obj.dataHora}, #{obj.paciente.nome} e #{obj.fisioterapeuta.nome}.

Para melhor compreensão do código, as linhas referentes à navegação, filtro, ordenação e paginação, conversor de data e hora e botões editar e excluir foram omitidos. No entanto, ao clicar no botão novo, por exemplo, o método novo( ) do *bean* gerenciado é invocado, e o EJB persiste uma nova agenda na base de dados.

A figura 29 mostra a continuação do código da página listar.xhtml onde foi colocado o diálogo. Para melhor visualização, também omitiu-se algumas linhas do código.

Figura 29 – Continuação do código da página listar.xhtml - dialog

```
        <ui:define name="dialogos">
            <p:dialog header="Edição" widgetVar="dlg" resizable="false" modal="true">
                <h:form id="formEdicao">
                    <p:panelGrid columns="4">
                        <f:facet name="footer">
                            <div align="center">
                                <p:commandButton
                                    value="Salvar"
                                    icon="ui-icon-disk"
                                    actionListener="#{controleAgenda.salvar()}"
                                    update=":formListagem :formEdicao"
                                    oncomplete="if(!args.validationFailed){PF('dlg').hide();}"/>
                            </div>
                        </f:facet>
                        <p:outputLabel value="Paciente" for="txtPaciente" />
                        <p:selectOneMenu ...8 linhas />
                        <p:outputLabel value="Data e Hora" for="txtDataHora"/>
                        <p:calendar ...6 linhas />
                        <p:outputLabel value="Fisioterapeuta" for="txtFisioterapeuta" />
                        <p:selectOneMenu ...8 linhas />
                        <p:outputLabel value="Funcionário" for="txtFuncionario" />
                        <p:selectOneMenu value="#{controleAgenda.objeto.funcionario}"
                                         id="txtFuncionario">
                            <f:converter converterId="converterFuncionario"/>
                            <f:selectItem noSelectionOption="true"
                                          itemLabel="Selecione um registro"/>
                            <f:selectItems value="#{controleAgenda.daoFuncionario.listObjects}"
                                           var="fu" itemLabel="#{fu.nome}"/>
                        </p:selectOneMenu>
                    </p:panelGrid>
                </h:form>
            </p:dialog>
        </ui:define>
    </ui:composition>
</html>
```

Fonte: Elaborada pelo Autor

O elemento <*p:dialog*> renderiza um diálogo na tela e, através dos atributos *resizable=”false”* e *modal=”true”*, configura-se incapaz de ser redimensionado e não pode-se alterar o foco do diálogo sem antes fechá-lo.

Dentro dele, foi colocado um formulário no qual serão informados os dados da consulta nos campos solicitados, por exemplo, o campo Funcionário é informado através de uma caixa de seleção <*p:selectionOneMenu*> a qual é construída através de um atributo daoFuncionario, método *listObjects*( ), do *bean* gerenciado. Já a data e hora agendadas são informadas através do elemento <*p:calendar*>.

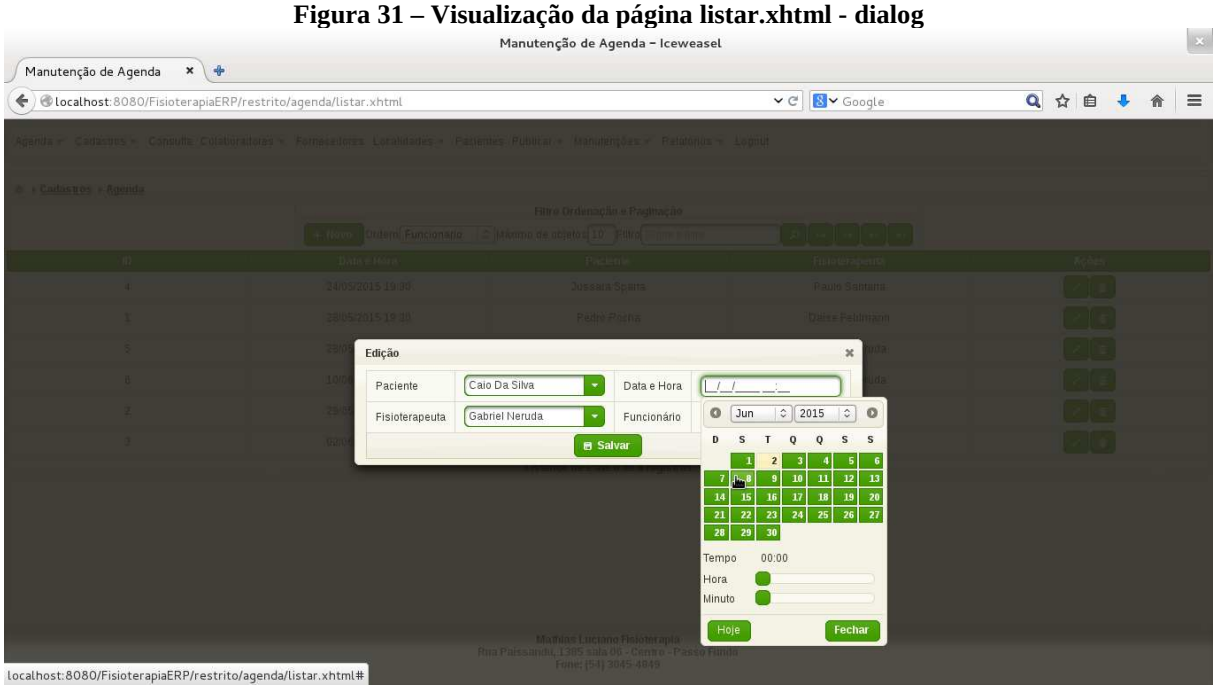
No rodapé do diálogo, foi colocado um elemento <*p:commandButton*> que, ao ser pressionado, invoca o método salvar( ) do *bean* gerenciado e, então, o EJB persiste a agenda na base de dados.

Nas figuras 30 e 31, são apresentadas as visualizações da página listar.xhtml implantada no servidor GlassFish.

Figura 30 – Visualização da página listar.xhtml - form

Fonte: Elaborada pelo Autor

Figura 31 – Visualização da página listar.xhtml - dialog



Fonte: Elaborada pelo Autor

4 CONSIDERAÇÕES FINAIS

Considerando a crescente demanda por serviços de saúde gerada pelo crescimento populacional, a melhoria na infraestrutura de redes computacionais nos últimos anos, e uma melhor gestão de recursos utilizados e serviços disponibilizados nas clínicas de saúde, um sistema de gestão web aparece como uma ferramenta poderosa para a administração.

O presente estudo teve como objetivo compreender os conceitos de orientação a objetos, bem como as tecnologias atuais de desenvolvimento de sistemas web com o uso da linguagem de programação Java, o padrão de projetos MVC, a API JavaServer Faces, o uso de JPA com o provedor de persistência Hibernate. Também, realizou-se estudo de conceitos de sistemas de banco de dados e, por fim, analise e desenvolvimento de um sistema de gestão para uma clínica de fisioterapia.

A implementação do sistema de gestão proposto foi facilitada pelo uso da plataforma Java EE. O aumento de produtividade é notável com ela, ao passo que em conjunto com a API JPA, o provedor de persistência Hibernate, os contentores EJB, o framework JSF e a API PrimeFaces abstraem uma série de serviços.

Embora o início do projeto tenha demandado muitas horas de estudo, a curva de aprendizado logo se equilibrou e o sistema foi se materializando. A finalização do projeto foi bastante motivadora, visto que a cada módulo terminado era possível visualizar os resultados e ter um *feedback* da clínica de fisioterapia.

A versão atual do sistema possui as quatro funções básicas (criação, leitura, atualização e exclusão) de persistência em banco de dados relacionais. Sendo assim, os módulos de cadastro de convênios, exames, funcionário, localidades, medicamentos e pacientes está funcional. Bem como, o módulo de agendamento de consultas, o pronturário eletrônico, os módulos de contas a pagar e receber. Como trabalhos futuros, pretende-se desenvolver a função de confirmação de consultas via SMS e e-mail, além de integrar a agenda do Google com a agenda de consultas do sistema, a fim de melhorar o serviço prestado aos pacientes da clínica de saúde.

REFERÊNCIAS

BARNES, David J.; KÖLLING, Michael.*Programação Orientada a Objetos com Java:* Uma introdução prática utilizando o Blue J. São Paulo: Pearson Prentice Hall, 2004.

BAUER, Christian; King, Gavin. *Java Persistence com Hibernate*. Rio de Janeiro: Ciência Moderna, 2007.

BEZERRA, Eduardo. *Princípios de Análise e Projetos de Sistemas com UML*. Rio de Janeiro: Elsevier, 2007.

DATE, Christopher J. *Introdução a Sistemas de Bancos de Dados*. Rio de Janeiro: Elsevier, 2003.

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

FURGERI, Sérgio. *Java 6*: Ensino Didático: Desenvolvendo e Implementando Aplicações. São Paulo: Érica, 2008.

GAMMA, Erich; HELM, Richard; JONSON, Ralph; VLISSIDES, John. *Padrões de Projeto*: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GOMES, Yuri Marx P. *Java na Web com JSF, Spring, Hibernate e Netbeans 6*. Rio de Janeiro: Ciência Moderna, 2008.

GONÇALVES, Antonio. *Introdução à Plataforma Java™ EE 6 com GlassFish™ 3*. Rio de Janeiro: Ciência Moderna, 2011.

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

GUEDES, Gilleanes T. A. *UML 2:* uma abordagem prática. São Paulo: Novatec, 2011.

HIDOCTOR, Software Médico Para Consultório. Disponível em: <http://www.hidoctor.com.br>. Acessado em: 15 out. 2014.

JENDROCK, Erik; CERVERA-NAVARRO, Ricardo; EVANS, Ian; HAASE, Kim; MARKITO, William. *Java Plataform, Enterprise Edition The Java EE Tutoria, Release 7* [Manual]. Redwood City, CA, USA. Disponível em: <https://docs.oracle.com/javaee/7/tutorial/index.html>. Acessado em: 10 jun. 2015.

MEDPLUS. Disponível em: <http://site.medplus.com.br>. Acessado em: 15 out. 2014.

MEDSYSTEM WEB, Sistema de Soluções para Gestão Médica de Clínicas e Consultórios Médicos. Disponível em: <http://medsystemweb.com.br>. Acessado em: 15 out. 2014.

NETO, Álvaro Pereira. *PostgreSQL*: Técnicas Avançadas: Versões Open Source 7.x e 8.x: Soluções para Desenvolvedores e Administradores de Banco de Dados. São Paulo: Érica, 2007.

PRESSMAN, Roger S. *Engenharia de Software:* uma abordagem profissional. Porto Alegre: AMGH, 2011.

TOTVS, Saúde. Disponível em: <http://www.totvs.com/software-de-gestao/saude>. Acessado em: 15 out. 2014.