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

**JHONATA ANTONIO ALVES FERREIRA**

# GESTÃO DO CONHECIMENTO WEB PARA ARMAZENAR RESOLUÇÕES DOS PROBLEMAS DE CLIENTES EM UM SUPORTE TÉCNICO

**Orientadora**
**Carmen Vera Scorsatto Brezolin**

**PASSO FUNDO**
**2017**

**JHONATA ANTONIO ALVES FERREIRA**

# GESTÃO DO CONHECIMENTO WEB PARA ARMAZENAR RESOLUÇÕES DOS PROBLEMAS DE CLIENTES EM UM SUPORTE TÉCNICO

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientadora: Carmen Vera Scorsatto Brezolin

**PASSO FUNDO**

**2017**

**JHONATA ANTONIO ALVES FERREIRA**

# GESTÃO DO CONHECIMENTO WEB PARA ARMAZENAR RESOLUÇÕES DOS PROBLEMAS DE CLIENTES EM UM SUPORTE TÉCNICO

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

Banca Examinadora:

_____

Carmen Vera Scorsatto Brezolin

_____

Rafael Marisco Bertei

_____

André Fernando Rollwagen

_____

Adilso Nunes de Souza

**PASSO FUNDO**

**2017**

**DEDICATÓRIA**

*À minha mãe Solange,*
*pelo carinho e incentivo a não desistir dos meus objetivos.*
*Ao meu pai Antônio (em memoria),*
*pela educação e ensinamento de que o caráter é o maior valor da vida.*
*À minha esposa Mary,*
*pelo amor, carinho e compreensão nas minhas decisões.*
*Aos professores,*
*pelos ensinamentos compartilhados.*
*À minha orientadora Carmen,*
*pelos conselhos, ensinamentos e motivação.*

## EPÍGRAFE

“Conquiste sua estrela
Sem tirar a estrela do ombro de seu colega. ”
Antônio Jesus Lopes Ferreira

## RESUMO

O setor de suporte técnico de uma empresa pode realizar diversas tarefas, em muitos casos os analistas atendem clientes em simultâneo, dificultando a agilidade e resolução dos problemas. Com base em estudos nos processos do setor de suporte técnico da empresa Software PF, foi identificado uma carência no armazenamento de informações relacionadas a resoluções dos problemas. Este projeto tem por objetivo desenvolver um sistema Web para auxiliar os atendimentos, padronizando as resoluções dos problemas de suporte técnico. O desenvolvimento do software foi realizado utilizando as tecnologias Java, JSF, JPA, banco de dados PostgreSQL, bibliotecas Hibernate e PrimeFaces. Após o desenvolvimento do sistema de gestão do conhecimento, o objetivo será alcançado, garantindo padronização e agilidade no atendimento ao cliente.



Palavras-chave: Gestão do Conhecimento. Sistema Web. Suporte Técnico. Java.

## ABSTRACT

The technical support sector of a company can perform a variety of tasks, in many cases the analysts serve customers simultaneously, making it difficult to solve problems. Based on studies in the processes of the technical support sector of the software company PF, a shortage was identified in the storage of information related to problem resolutions. The purpose of this project is to develop a Web system to assist the attendants by standardizing the resolutions of technical support problems. Software development was performed using Java, JSF, JPA, PostgreSQL database, Hibernate and PrimeFaces libraries. After the development of the knowledge management system, the goal will be achieved, guaranteeing standardization and agility in customer servisse.



Keywords: Knowledge Management. Web System. Technical Support. Java.

## LISTA DE FIGURAS

Figura 1 - Ambiente de desenvolvimento Java ..... 15
Figura 2 - Relacionamento hierárquico entre métodos ..... 19
Figura 3 - Hierarquia de herança dos membros da comunidade universitária ..... 19
Figura 4 - Arquitetura em camadas - persistência ..... 22
Figura 5 - Consulta em bancos de dados diferentes ..... 24
Figura 6 - Consulta JPQL ..... 24
Figura 7 - Persistence.xml ..... 26
Figura 8 - Propriedade create ..... 26
Figura 9 - MVC no JSF ..... 28
Figura 10 - Limite de valores ..... 30
Figura 11 - Diagrama de Casos de Uso ..... 34
Figura 12 - Diagrama de Classes ..... 36
Figura 13 - IDE NetBeans versão 8.1 ..... 39
Figura 14 - Servidor GlassFish versão 4 ..... 40
Figura 15 - PostgreSQL e PGAdmin III ..... 41
Figura 16 - Estrutura do Sistema ..... 42
Figura 17 - Bibliotecas utilizadas ..... 43
Figura 18 - Trecho do código da classe Conhecimento ..... 44
Figura 19 - Estrutura da Camada Modelo ..... 45
Figura 20 - Trecho do código do arquivo persistence.xml ..... 46
Figura 21 - Trecho do Código da classe DAOGenerico ..... 47
Figura 22 - Código da classe ConhecimentoDAO ..... 47
Figura 23 - Código da classe ControleConhecimento ..... 48
Figura 24 - Estrutura da Camada de Visão ..... 49
Figura 25 – Código do menu do sistema ..... 50
Figura 26 - Pagina Inicial do Sistema de Gestão do Conhecimento ..... 51
Figura 27 - Validação de acesso ao Sistema ..... 52
Figura 28 - Pagina inicial do perfil administrador ..... 52
Figura 29 - Página inicial perfil restrito ..... 53
Figura 30 - Listagem dos Conhecimentos ..... 54
Figura 31 - Pesquisa de Conhecimentos ..... 55
Figura 32 - Cadastrando um novo conhecimento ..... 55

Figura 33 - Removendo um conhecimento ..... 56
Figura 34 - Saindo do Sistema ..... 56

## LISTA DE ABREVIATURAS E SIGLAS

AJAX – (Asynchronous Javascript And XML), Javascript Assíncrono e XML

API – (Application Programming Interface), Interface de Programação de Aplicativos

DAO – (Data Acces Object), Objeto de Acesso a Dados

IDE – (Integrated Developmente Environment), Ambiente Integrado de Desenvolvimento

JDK – (Java Developmente Kit), Kit de Desenvolvimento Java

JIT – (Just-In-Time), Na Hora Certa

JPA – (Java Persistence API), API de Persistência Java

JPQL – (Java Persistence Query Language)

JSF – (JavaServer Faces)

JVM – (Java Virtual Machine), Máquina Virtual Java

MVC – (Model – View - Controller), Modelo – Visão - Controle

ORM - Mapeamento Objeto Relacional

SQL – (Structure Query Language), Linguagem de Consulta Estruturada

UI – (User Interface), Interface do Usuário

VM – (Virtual Machine), Máquina Virtual

UML – (Unified Modeling Language)

SGBD – Sistema de Gerenciamento de Banco de Dados

## SUMÁRIO


1 INTRODUÇÃO ..... 11

1.1.1. Objetivo geral ..... 12
1.1.2. Objetivos específicos ..... 12

2 REFERENCIAL TEÓRICO ..... 13
2.1 TECNOLOGIAS USADAS ..... 13
2.1.1 História do Java ..... 13
2.1.2 Classe Java ..... 14
2.1.3 Ambiente de desenvolvimento Java ..... 14
2.1.4 Módulos de Programa ..... 17
2.1.5 Superclasse e sunclasses ..... 19
2.1.6 Polimorfismo ..... 20
2.1.7 Persistência de dados ..... 21
2.1.7.1 Camadas de persistência ..... 21
2.1.7.2 ORM ..... 23
2.1.7.3 JPA ..... 23
2.1.7.3.1 Padrão DAO ..... 25
2.1.7.4 Hibernate ..... 25
2.1.7.4.1 Automatizar a criação e evolução das tabelas do banco de dados ..... 26
2.1.8 JSF ..... 27
2.1.9 MVC ..... 28
2.1.10 Primefaces ..... 29
2.1.11 PostgreSQL ..... 30
2.1.12 Gestão do conhecimento ..... 31
2.1.13 UML ..... 31

3 METODOLOGIA ..... 33
3.1 ESTUDO DO SISTEMA EXISTENTE ..... 33

4 MODELAGEM DO SISTEMA ..... 34
4.1 DIAGRAMA DE CASOS DE USO DA GESTÃO DO CONHECIMENTO ..... 34
4.2 DIAGRAMA DE CLASSES DA GESTÃO DO CONHECIMENTO ..... 36

5 DESENVOLVIMENTO ..... 39
5.1 AMBIENTE DE DESENVOLVIMENTO ..... 39
5.1.1 IDE ..... 39
5.1.2 Servidor e SGBD ..... 39
5.1.3 Estrutura do sistema ..... 41
5.1.4 Bibliotecas ..... 42
5.1.5 Camada modelo ..... 43
5.1.5.1 Classes DAO ..... 46
5.1.6 Camada de controle ..... 47
5.1.7 Camada de visão ..... 49

6 RESULTADOS ..... 51

7 CONSIDERAÇÕES FINAIS ..... 57

REFERÊNCIAS ..... 58

# 1 INTRODUÇÃO

Algumas empresas poderão ter a necessidade de utilizar um sistema para gerenciamento de conhecimentos, auxiliando os colaboradores a terem um melhor rendimento sobre as tarefas que exercem. Um sistema de gestão do conhecimento pode auxiliar esse processo da melhor forma possível, pois trata diretamente na organização interna de uma empresa. Uma organização que possui um sistema de gestão do conhecimento poderá manter um nível de satisfação de seus clientes mais elevado, pois prevê que os atendimentos possam ser acompanhados em tempo hábil, conforme as políticas e processos da empresa.

A internet é um meio de comunicação que se popularizou, tendo isso como base, um sistema de gestão do conhecimento Web possibilitará que uma organização possa gerenciar seus conhecimentos[1] internamente e externamente.

Facilidade no acesso ao sistema é um dos principais pontos positivos em uma aplicação Web. Outro fator importante é de não exigir computadores com hardwares específicos, pois o sistema fica disponível para seu usuário através do seu navegador (*Browser*).

## 1.1. MOTIVAÇÃO

Uma empresa que possui um fluxo intenso de atendimentos e que não possui um sistema de gestão do conhecimento pode ter muitos problemas de relacionamento com seus clientes.

Quando um cliente solicita um atendimento, o mesmo deseja ser bem atendido, e principalmente, de forma ágil. Um cliente que retorna solicitando suporte reportando o mesmo problema tem em mente que, se já foi resolvido uma vez, a próxima será mais rápido a resolução, independente do horário ou atendente que irá realizar o suporte. Porém, se a empresa não possuir um sistema de gestão do conhecimento para gerenciar os conhecimentos, é provável que o problema do cliente não seja resolvido de forma ágil, mesmo sendo o mesmo analista que o atendeu anteriormente, pois se o fluxo de atendimentos for alto, é possível que o analista não se recorde de como resolveu o problema.

---

[1] Gerenciar informações dos problemas de clientes em um suporte técnico.

Um sistema de gestão do conhecimento pode trazer muitos benefícios para a empresa e seus clientes. Quando é reportado um problema e o mesmo é solucionado de forma rápida, o cliente ficará satisfeito, mas se solicitar suporte e precisar ficar esperando por muito tempo até o problema ser resolvido, é muito provável que o cliente não ficará satisfeito e em muitos casos, solicitar o cancelamento do serviço adquirido.

Nesse contexto buscou-se analisar como um sistema de gestão de conhecimento pode auxiliar no gerenciamento de uma empresa?

### 1.1.1. Objetivo geral

Desenvolver um sistema de gestão do conhecimento Web, que possibilite a inserção, armazenamento e consulta dos conhecimentos para a resolução dos problemas de clientes em um suporte técnico.

### 1.1.2. Objetivos específicos

Desenvolver um sistema Web, para que futuramente seja implantado na empresa Software PF[2].

Possibilitar o gerenciamento de conhecimentos.

Programar o acesso ao sistema por meio de autenticação, com níveis de perfis e usuários ativos.

[2] Empresa de nome fictício para preservar a integridade da empresa verdadeira.

# 2 REFERENCIAL TEÓRICO

Neste capítulo serão descritas as tecnologias utilizadas para a construção do software proposto, bem como os diagramas elaborados.

## 2.1 TECNOLOGIAS USADAS

Nesta seção serão apresentados os assuntos referentes ao Java, JPA, JSF, Hibernate, Primefaces, PostgreSQL, Glassfish e sistema de gestão de conhecimento.

### 2.1.1 História do Java

Em 1991 a empresa Sun Microsystems, realizou um financiamento de um projeto de pesquisa corporativa interna, o resultado desse projeto de pesquisa foi o surgimento da linguagem Oak baseada em C++, seu criador James Gosting derivou o nome da linguagem em homenagem a uma espécie de árvore de carvalho vista por sua janela na Sun Microsystems. Mais tarde foi descoberto que já existia uma linguagem de computador com o nome de Oak. O nome Java (cidade de origem de um determinado café importado) surgiu após uma equipe da Sun visitar uma cafeteria local (DEITEL, 2010).

A Sun Microsystems estava prevendo um grande desenvolvimento de mercado para dispositivos eletrônicos inteligentes destinados ao consumidor final, porém esse crescimento não foi como a Sun estava prevendo e o seu projeto de pesquisa passou por muitas dificuldades. Em 1993 a Web começou a entrar em popularidade e a Sun aproveitou para adicionar conteúdo dinâmico as páginas Web com o Java, como animações e interatividade (DEITEL, 2010).

O Java foi formalmente anunciado pela Sun Microsystens em maio de 1995 em uma conferência do setor. Para Deitel (2010, p.6):

> O Java é agora utilizado para desenvolver aplicativos corporativos de grande porte, aprimorar a funcionalidade de servidores da Web (os computadores que fornecem o conteúdo que vemos em nossos navegadores da Web), fornecer aplicativos para dispositivos voltados para o consumidor popular (como telefones celulares, pagers e PDAs) e para muitos outros propósitos.

A comunidade de negócios foi atraída por causa do grande interesse que o Java demonstrou na Web (DEITEL, 2010).

Java se destaca por ser uma linguagem portável para outros sistemas operacionais, com o passar dos anos o Java foi se amadurecendo e atualmente pode ser encontrado em desktop, web e aplicativos móveis. Segundo Gonçalves (2007, p. 8) “milhões de pessoas já aprenderam essa linguagem e, grandes empresas a estão usando. Lugares como a NASA, IBM, ESPN entre outros são apenas exemplos da confiabilidade que a linguagem Java demonstra em seus utilizadores”.

### 2.1.2 Classe Java

As classes são partes que consistem um programa em Java. Classes possuem métodos que são utilizados para realizar tarefas e devolver informações depois que forem finalizadas.

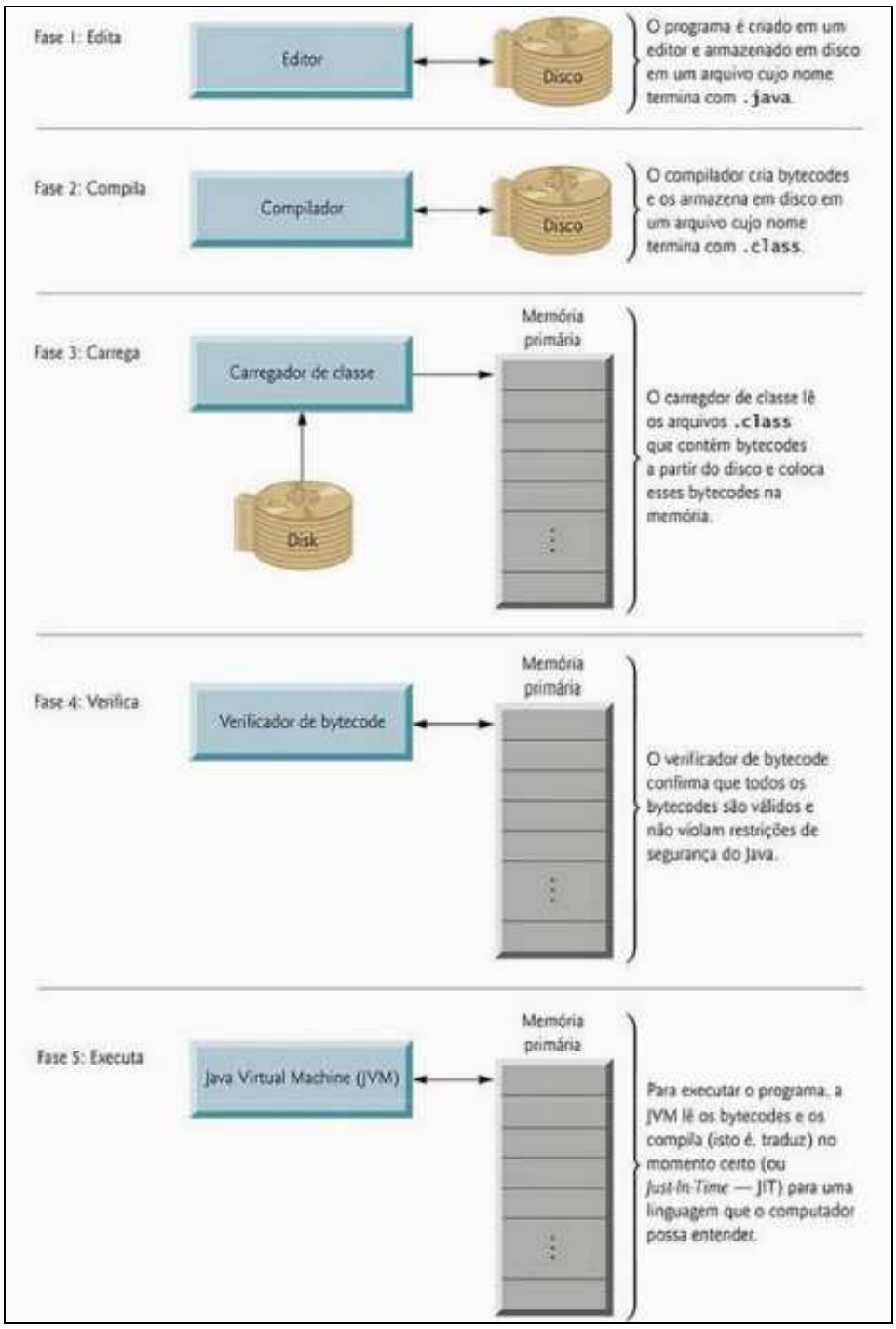
> Você pode criar cada parte necessária para formar seus programas Java. Entretanto, a maioria dos programadores Java tira proveito das ricas coleções de classes existentes nas bibliotecas de classe Java, que também são conhecidas como Java APIs (Application Programming Interfaces) (DEITEL, 2010, p. 7).

Existem dois modos para aprender Java. Primeiro é a própria linguagem Java que é de o programador mesmo desenvolver suas classes e o segundo é de utilizar classes nas extensões de bibliotecas Java (DEITEL, 2010).

### 2.1.3 Ambiente de desenvolvimento Java

Os programas desenvolvidos em Java possuem cinco fases, que são a edição, compilação, carregamento, verificação e execução (ilustrado na Figura 1).

**Figura 1 - Ambiente de desenvolvimento Java**




**FONTE: DEITEL, 2010**

Na primeira fase, é editado um arquivo utilizando um programa editor, é digitado um programa Java realizando as devidas alterações e salvo em uma unidade de disco. O arquivo que contém o código-fonte Java fica armazenado com a extensão .java (DEITEL, 2010).

Para ambiente Linux, pode se usar o editor vi, e no Windows o notepad, para uma melhor visualização do código-fonte, pode se instalar no Windows o notepad++.

A forma mais comum de se criar um programa em Java é utilizando um ambiente de desenvolvimento integrado (*Integrated Development Environment - IDE*), que disponibilizam ferramentas para o desenvolvimento de software. Está incluído nas IDEs os editores para escrever e editar o código-fonte e depuradores que consistem em localizar no código-fonte algum erro de lógica (DEITEL, 2010).

Na próxima fase é utilizado o comando javac para compilar um programa, se o nome do programa fosse helloworld.java o comando completo a ser digitado ficaria da seguinte forma:

**Javac helloworld.java**

Este comando é executado no prompt de comando do Windows ou no prompt de Shell no Linux ou aplicativo terminal do Mac OS X. Após a execução o compilador cria um arquivo .class chamado helloworld,class com a versão compilada do programa. Depois de compilado o código-fonte é convertido em bytecodes, segundo Deitel (2010, p. 10):

> Os bytecodes são executados pela Java Virtual Machine (JVM) – uma parte do JDK é a base da plataforma Java. A máquina virtual (Virtual Machine - VM) é um aplicativo de software que simula um computador, mas oculta o sistema operacional e o hardware subjacentes dos programas que interagem com ela. Se a mesma VM for implementada nas várias plataformas de computador, os aplicativos que ela executa podem ser utilizados em todas essas plataformas.

Os bytecodes são independentes de plataforma, não necessitam de um hardware em específico, pois não se trata de uma linguagem de máquina. O bytecode roda em uma JVM (Máquina Virtual Java), nesse caso o mesmo bytecode pode rodar em diversas plataformas, precisa apenas conter uma JVM que atende a versão do Java que os bytecodes foram compilados (DEITEL, 2010).

Na terceira fase, o programa em Java que vai ser executado é armazenado em memória pela JVM, conforme argumenta Deitel (2010, p. 10) ”O carregamento de classe da JVM pega os arquivos .class que contém os bytecodes do programa e transfere-os para a memória primaria. O carregador de classe também carrega qualquer arquivo .class fornecido pelo Java que seu programa utiliza.” A partir de um

disco em seu sistema ou em uma rede, os arquivos com extensão .class podem ser carregados.

Na quarta fase, os bytecodes são analisados pelo verificador de bytecode, para assegurar que são válidos e se não violam as regras de segurança do Java. Os programas que chegam pela rede recebem uma forte certificação de segurança do Java, isso impede que os arquivos ou o sistema fique danificado (DEITEL, 2010).

Na última fase, as ações especificadas pelo programa Java são realizadas após a JVM executar os bytecodes do programa. Os programas Java que eram executados nas primeiras versões da JVM ficavam lentos, pois era interpretado um bytecode por vez.

> As JVMs atuais executam bytecodes utilizando uma conbinação de interpretação e a chamada compilação Just-In-Time (JIT). Nesse processo, a JVM analisa os bytecodes á medida que eles são interpretados, procurando hot spots (pontos ativos) – partes dos bytecodes que executam com freqüência. Para essas partes, um compilador Just-In-Time (JIT) – conhecido como compilador Java HotSpot – traduz os bytecodes para a linguagem de máquina de um computador subjacente. Quando a JVM encontra novamente essas partes compiladas, o código de linguagem de máquina mais rápido PE executado. Por tanto, os programas Java na realizade passam por duas fazes de compilação – uma em que o código-fonte é traduzido em bytecodes (para a portabilidade entre JVMs em diferentes plataformas de computador) e uma segunda em que, durante a execução, os bytecodes são traduzidos em linguagem de máquina para o computador real em que o programa é executado. (DEITEL, 2010, p. 10).

O funcionamento de um programa em Java depende de todas as fases, o programa que contém alguma falha em um determinado ponto de seu código-fonte pode não funcionar na primeira tentativa de execução, “um programa executável talvez tente realizar uma operação de divisão por zero (uma operação ilegal para a aritmética de número inteiro em Java). Isso faria o programa Java imprimir uma mensagem de erro. Se isso ocorresse, você teria de retornar à fase de edição” (DEITEL, 2010, p. 10).

### 2.1.4 Módulos de Programa

A linguagem Java contém três módulos, os métodos, classes e pacotes. Possui combinações que os desenvolvedores vão montando no momento que estão escrevendo o programa, podem ser realizadas as combinações utilizando métodos e classes disponíveis na Java *Application Programming Interface (*conhecida como

Java API ou biblioteca de classes Java*)* e nas demais bibliotecas de classes Java disponíveis. Para que as classes relacionadas possam ser reutilizadas, elas ficam agrupadas em pacotes e depois devem ser importadas para o programa, segundo Deitel (2010, p.155):

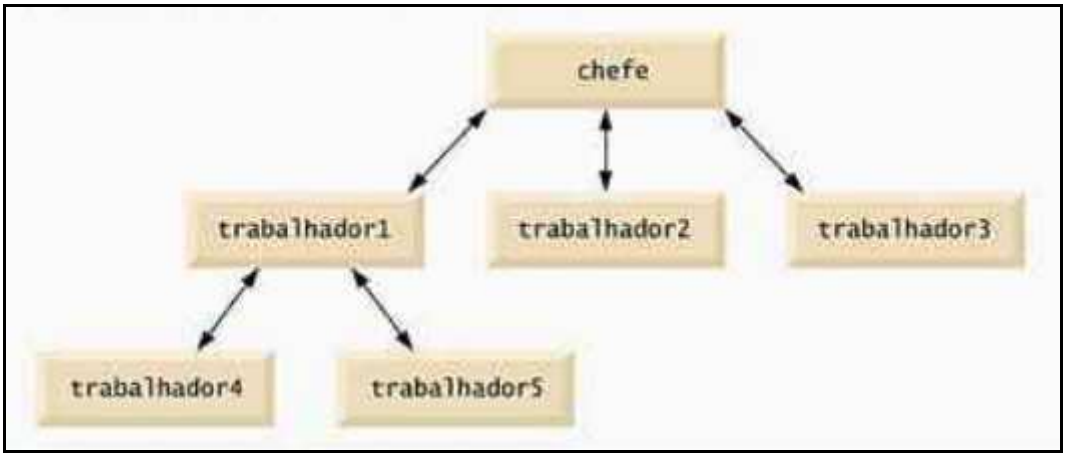
> A Java API fornece uma rica coleção de classes predefinidas que contém métodos para realizar cálculos matemáticos comuns, manipulações de string, manipulação de caractere, operação de entrada/saída, operações de banco de dados, operações de rede, processamento de arquivo, verificação de erros e muitas outras tarefas úteis.

Os métodos que são chamados também de funções ou procedimentos em determinadas linguagens, auxiliam para desenvolver um sistema modular colocando as tarefas em unidades auto-contidas. As instruções declaradas nos métodos são informadas apenas uma única vez e podem ser reutilizadas em um programa a partir de várias localizações.

A possibilidade de utilização de métodos já existentes incentiva o programador a desenvolver um sistema modular, tornando o desenvolvimento do programa mais gerenciável e mais fácil de ser depurado, pois são criados a partir de peças menores e mais simples (DEITEL, 2010).

Os métodos podem ser escritos de uma forma hierárquica, podendo um método invocado realizar uma chamada para outro método. Conforme pode ser visto na figura 2, o método chefe realiza uma chamada para o método trabalhador1 que por sua vez chama os métodos trabalhador4 e trabalhador5 de forma oculta, segundo Deitel (2010, p. 156) ”Esse “ocultamento” dos detalhes de implementação promove a boa engenharia de software”.

**Figura 2 - Relacionamento hierárquico entre métodos**

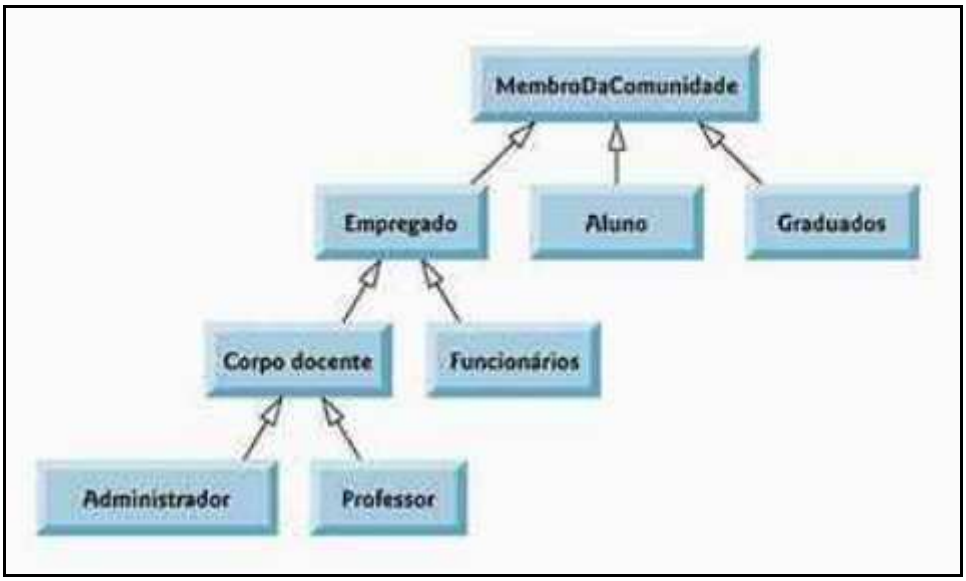



**FONTE: DEITEL, 2010**

### 2.1.5 Superclasse e sunclasses

Um objeto de uma determinada classe também pode ser um objeto de outra classe, essa estrutura de relação de herança é parecida com uma árvore. A superclasse possui um relacionamento hierárquico com suas subclasses. Conforme ilustrado na figura 3, diversos membros fazem parte de uma comunidade universitária, os membros são empregados, alunos e graduados. Os funcionários e o corpo docente são empregados da comunidade universitária. Os membros do corpo docente são os professores e administradores (DEITEL, 2010).

**Figura 3 - Hierarquia de herança dos membros da comunidade universitária**




**FONTE: DEITEL, 2010**

De maneira semelhante os objetos de subclasse e objetos de superclasses podem ser tratados. Conforme declara Deitel (2010, p. 281) “Os objetos de todas as classes que estendem uma superclasse comum podem ser tratados como objetos dessa superclasse”.

Um método de uma subclasse pode ter sido herdado, porém sem haver necessidade ou até mesmo não deveria estar na subclasse, o método de uma subclasse precisa ser adequado e com frequência ser personalizado, esse é um exemplo de problemas que se pode ter com a utilização de herança (DEITEL, 2010).

Não são todas as classes que se relacionam de forma a ser um relacionamento de herança, segundo Deitel (2010, p. 280):

> Esses relacionamentos criam classes compondo classes existentes. Por exemplo, considerando as classes Empregado, DataDeNascimento e NúmeroDeTelefone, não é correto dizer que um Empregado é uma DataDeNascimento ou que um Empregado é um NúmeroDeTelefone. Entretando, um Empregado tem uma DataDeNascimento e um Empregado tem um NúmeroDeTelefone.

### 2.1.6 Polimorfismo

O polimorfismo permite que objetos compartilhem a mesma superclasse, podendo esse acesso ser direto ou indiretamente, isso facilita a programação, pois os objetos são tratados como se fossem da superclasse. Novas classes podem ser inseridas com nenhuma ou pouca alteração, porém as novas classes precisam fazer parte da hierarquia de herança processado pelo programa genericamente (DEITEL, 2010).

Apenas se altera as classes que necessitam de conhecimento direto das novas classes que foram adicionadas à hierarquia. Conforme exemplo citado por Deitel (2010, p. 305):

> Se estendermos a classe Animal para criar a classe Tartaruga (que poderia responder a uma mensagem mover deslizando uma polegada), precisaremos escrever somente a classe Tartaruga e a parte da simulação que instancia um objeto Tartaruga. As partes da simulação que processam cada Animal genericamente podem permanecer idênticas.

### 2.1.7 Persistência de dados

Grande parte dos sistemas desenvolvidos precisam realizar a persistência dos dados, no desenvolvimento de aplicações a persistência de dados é um conceito fundamental. Uma aplicação de informação prática e funcional precisa realizar essa persistência, pois se o sistema perde os dados ao ser finalizado, o mesmo já não seria mais prático e funcional. Quando se fala persistir dados em Java, isso quer dizer sobre como armazenar os dados em um banco de dados relacional utilizando o SQL (BAUER; KING, 2007).

#### 2.1.7.1 Camadas de persistência

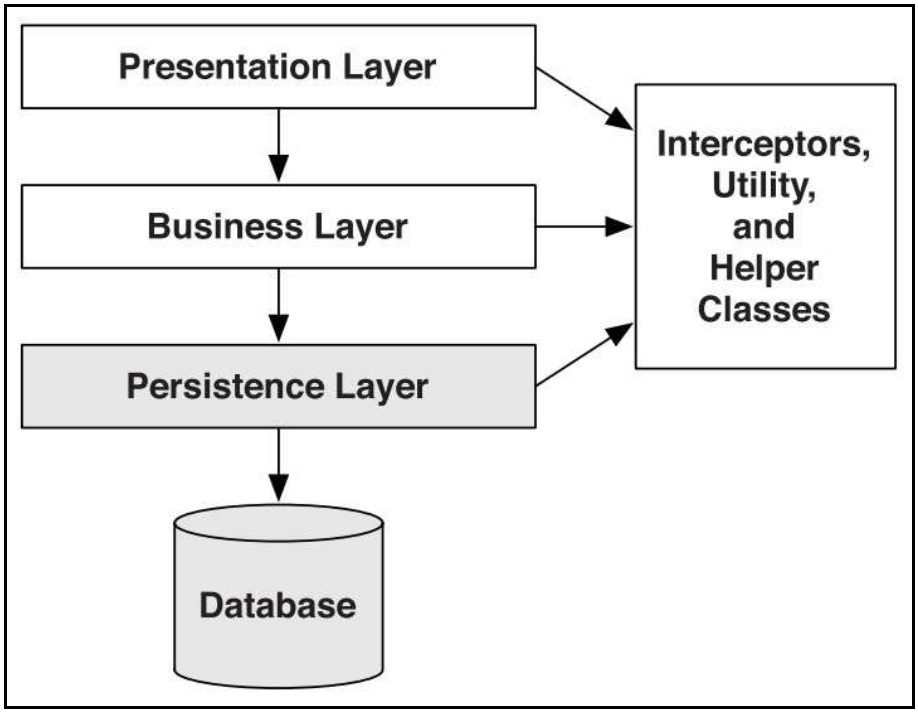
Normalmente as classes são organizadas por funcionalidade em aplicações de médio ou grande porte. Camadas de códigos que representam as funcionalidades são incluídas em uma arquitetura típica de orientação a objetos. Em uma arquitetura de sistemas em camada, uma boa prática é organizar as classes e componentes responsáveis pela persistência, agrupando separadamente das demais classes.

Em uma arquitetura em camadas, é permitido que uma funcionalidade seja implementada sem afetar o código-fonte das demais camadas. Os tipos de dependências são determinados pela distribuição em camadas, Segundo Bauer e King (2007, p. 20):

> As camadas se comunicam do topo para a base. Uma camada é dependente somente da camada abaixo dela.
>
> Cada camada não sabe de qualquer outra camada, exceto a logo abaixo dela.

Um sistema de alto nível utiliza camadas diferentes, uma para apresentação, outra para a lógica, e por fim uma para a persistência como ilustrado na Figura 4.

**Figura 4 - Arquitetura em camadas - persistência**




**FONTE: (BAUER; KING, 2007)**

- Camada de apresentação *'Presentation Layer'* – O código-fonte responsável pela apresentação, navegação na tela e controle da página ficam na camada de apresentação. A lógica que fica mais ao topo é a da interface com o usuário (BAUER; KING, 2007).
- Camada de negócio *'Business Layer'* – A camada de negócio pode variar conforme a aplicação implementada, geralmente essa camada fica de responsável para aplicar qualquer regra de negócio ou necessidades da aplicação, como parte do problema entendidos pelos usuários. É incluído nessa camada um código para chamar a regra de negócio mais adequada à situação. Em determinados sistemas, essa camada reaproveita o modelo que a camada de persistência definiu ou possui uma representação própria interna das entidades de negócio e domínio (BAUER; KING, 2007).
- Camada de persistência *'Persistence Layer'* – Os dados que são armazenados e recuperados de um repositório, é realizado por um grupo de classes e componentes da camada de persistência. São incluídos nessa camada os modelos de entidades de negócio e domínio (BAUER; KING, 2007).

- Banco de dados *'Database'* – A representação persistente do sistema é o banco de dados, que fica fora da aplicação, para Bauer e King (2007, p. 21) "Se um banco de dados SQL é utilizado, o banco de dados inclui um esquema relacional e possivelmente procedimentos armazenados".
- Classes de ajuda e de utilidade *'Interceptors, Utility, and Helper Classes*' – Todas as camadas da aplicação utilizam classes de ajuda e de utilidade (como classes *Exception* para tratamento de erro). Essas classes fazem parte de um conjunto de infra-estrutura. Como as regras de dependência entre as camadas da arquitetura em camadas não são obedecidas pelos elementos de infra-estrutura, eles não podem formar uma camada (BAUER; KING, 2007).

#### 2.1.7.2 ORM

ORM (mapeamento objeto/relacional) é a persistência transparente e automatizada em uma aplicação Java dos objetos para as tabelas de um banco de dados relacional, para realizar o mapeamento entre o banco de dados e os objetos são utilizados metadados que descrevem esse processo, conforme cita Bauer e King (2007, p. 25):

> ORM, em sua essência, trabalha com a transformação (reversível) dos dados de uma representação para outra. Isso implica algumas penalidades de performance. Contudo, se o ORM é implementado como um middleware, existem várias oportunidades para a otimização que não iria existir para uma camada de persistência codificada à mão. O provimento e o gerenciamento, mas o custo é menor do que o custo equivalente envolvido na manutenção de uma solução codificada a mão. (E até mesmo bancos de dados de objetos necessitam de quantias significativas de metadados.).

#### 2.1.7.3 JPA

O JPA (*Java Persistence API*) é uma API da linguagem Java para realizar a persistência dos dados, é um conjunto de normas, regras e interfaces que definem uma especificação. Para que uma especificação possa ser utilizada em um projeto Java, precisa-se de uma implementação. Segundo Oliveira (2013, p. 5) "No caso da JPA, a implementação mais famosa e utilizada no mercado é o Hibernate, mas

existem mais no mercado como OpenJPA, EclipseLink, Batoo e outras. Cada implementação tem suas vantagens na hora da utilização".

A grande facilidade da JPA é a possibilidade de realizar portabilidade. Para isso junto com a JPA vem a linguagem JPQL (*Java Persistence Query Language*). A JPQL possui uma sintaxe que se assemelha a uma query normal (CORDEIRO, 2012).

Uma simples consulta pode ser escrita em diversas formas, variando conforme o banco de dados que está sendo utilizado (ilustrado na Figura 5).

**Figura 5 - Consulta em bancos de dados diferentes**

```
# MySQL
select * from cachorros LIMIT 10;

# Postgres
select * from cachorros LIMIT 10;

# MS SQLServer
select top 10 * from cachorros;

# Oracle
select * from cachorros where rownum <= 10;
```

**FONTE: OLIVEIRA, 2012**

O código para realizar uma consulta utilizando o JPQL é relativamente simples (ilustrado na Figura 6).

**Figura 6 - Consulta JPQL**

```
//Variável consulta está recebendo os valores do select
//realizado utilizando o padrão "JPQL"
String consulta = "select p from Projeto p";
```

**FONTE: DO AUTOR**

A vantagem de utilizar o JPQL é de não escrever uma consulta específica para um determinado banco de dados, não se informa o caractere * (asterisco), apenas informa o que foi atribuído aos objetos, nesse caso a letra p é um *alias*

(apelido) dos objetos do tipo Projeto (OLIVEIRA, 2013). Conforme cita o autor, “A JPQL na verdade é uma linguagem baseada em objetos. Ao invés de descrevermos como ficará a ligação das tabelas em uma query, escrevemos como os objetos se relacionam” (2013, p. 101). A sintaxe é escrita na query conforme está escrito as classes e seus atributos.

#### 2.1.7.3.1 Padrão DAO

Quando não se utiliza uma ferramenta de mapeamento objeto-relacional, o desenvolvedor precisa manipular um grande volume de código, realizar tratamentos de exceções, SQL, abrir e fechar conexões, entre outras tarefas. Para melhorar a legibilidade e separar a responsabilidade, toda a lógica que contém informações de acesso a dados é inserida dentro de uma classe em especifico que se chama DAO (*Data Acces Object*). A classe DAO fica responsável de prover para o sistema todas as operações de acesso a dados, como a inserção, exclusão, leitura e alteração (CORDEIRO, 2012).

Boas práticas de programação como programar orientado a interfaces e não a implementação, já que a implementação fica separada do código e para as operações de dados se utiliza uma interface. Utilizando essa forma de programar, se pode trocar de implementação sem alterar o código. Segundo Cordeiro (2012, p. 66) “A flexibilidade é tanta que no caso mais simples podemos ter implementações do mesmo DAO para bancos diferentes ou para mecanismos diferentes de bancos relacionais, como os bancos NoSQL ou mesmo arquivos texto”.

### 2.1.7.4 Hibernate

Os códigos escritos em SQL são específicos de cada banco de dados. O Hibernate, implementação da JPA oculta esses detalhes de cada banco de dados. Por funcionalidade o Hibernate tem a abstração do banco de dados, se for realizada uma troca de banco de dados, não é necessário alterar o código, pois as definições de cada banco ficam armazenadas no arquivo *persistence.xml* (CORDEIRO, 2012) (ilustrado na Figura 7).

**Figura 7 - Persistence.xml**

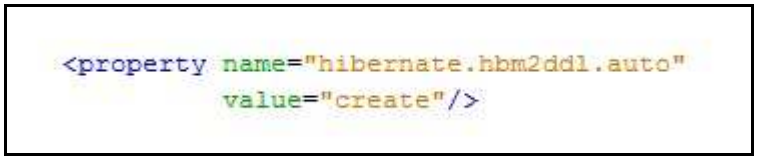
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="Default" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.cache.provider_class"
                value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.connection.username"
                value="postgres"/>
      <property name="hibernate.connection.driver_class"
                value="org.postgresql.Driver"/>
      <property name="hibernate.connection.password"
                value="postgres"/>
      <property name="hibernate.connection.url"
                value="jdbc:postgresql://localhost:5432/default"/>
      <property name="hibernate.hbm2ddl.auto"
                value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

**FONTE: DO AUTOR**

### 2.1.7.4.1 Automatizar a criação e evolução das tabelas do banco de dados

Quando se manipula um banco de dados, é preciso estar com as tabelas criadas e atualizadas, se o projeto estiver no começo, não terá nenhuma tabela criada. Para simplificar esse processo, o Hibernate cria as tabelas no banco de dados, apenas precisa realizar a configuração de qual estratégia deverá ser utilizada, por meio da propriedade name="hibernate.hbm2ddl.auto". O Hibernate cuida para que as tabelas do banco de dados sejam excluídas e recriadas, utilizando a propriedade *create* (ilustrado na Figura 8), dessa forma, irá manipular a partir de um banco de dados vazio (CORDEIRO, 2012).

**Figura 8 - Propriedade create**



```
<property name="hibernate.hbm2ddl.auto"
          value="create"/>
```

**FONTE: DO AUTOR**

Pode se informar mais três possibilidades para essa propriedade, conforme cita Cordeiro (2012, p. 33):

> - Create-drop – as tabelas são criadas pelo próprio Hibernate e excluídas somente no final da execução do programa caso solicitado pelo desenvolvedor;
> - Update – nada é excluído, apenas criado, ou seja, todos os dados são mantidos. Útil para aplicações que estão em produção, das quais nada pode ser apagado;
> - Validate – não cria e nem exclui nada, apenas verifica se as entidades estão de acordo com as tabelas e, caso não esteja, uma exceção é lançada.

### 2.1.8 JSF

O JSF (*Java Server Faces*) é um framework da linguagem Java baseado no padrão MVC (Modelo – Visão - Controle), com ele é possível desenvolver interfaces de usuários web colocando em um formulário os componentes e depois os liga a objetos Java, sem ter a necessidade de misturar código. Uma das facilidades do JSF está em seu modelo de componentes extensíveis e possui um grande número de componentes de terceiros. O JSF permite a separação entre gerenciamento de configurações, conexões com serviços externos, navegação, lógica de negócios e de apresentação (GEARY; HORSTMANN, 2007).

As seguintes partes compõem o JSF:

- Interface de usuário, que é um grupo de componentes pré-fabricados de UI;
- Possui um modelo de programação que é orientado a eventos;
- Possui um modelo de componentes que possibilita o fornecimento de componentes adicionais de desenvolvedores independentes (GEARY; HORSTMANN, 2007).

Campos de *input* e botões são componentes do JSF relativamente simples, tabelas e árvores por outro lado já são bastante sofisticados.

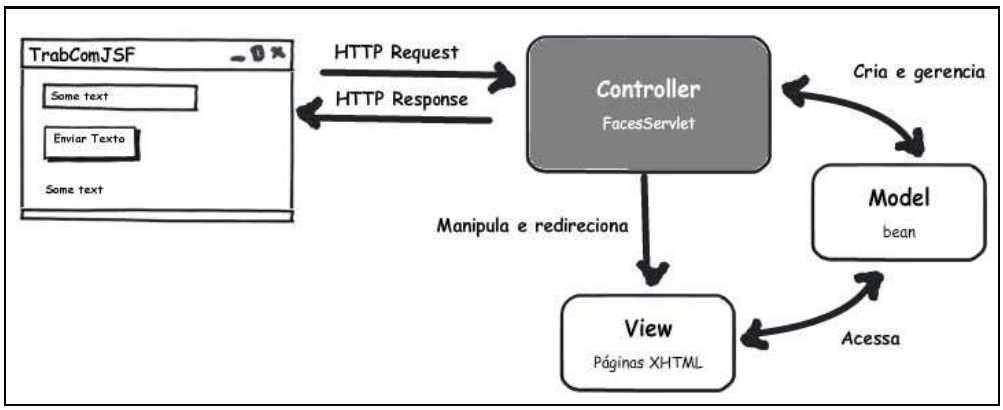
Todos os códigos necessários para realizar a organização de componentes e manipulação de eventos estão contidos no JSF, os desenvolvedores de aplicativos se preocupam apenas com a lógica de programação, ignorando os detalhes de

manipulação de eventos e organização de componentes (GEARY; HORSTMANN, 2007).

### 2.1.9 MVC

O Modelo Visão e Controle mais conhecido pela sigla MVC (*Model – View - Controller*) é um conceito de design e desenvolvimento utilizado para separar em três partes distintas uma determinada aplicação (ilustrado na Figura 9), a parte do trabalho atual administrada pela aplicação está relacionado o Modelo, a Visão é responsável por exibir as informações da aplicação e o Controle gerencia o Modelo e a Visão, coordenando a melhor forma de execução do trabalho e exibindo a interface correta (GOLÇALVES, 2007).

**Figura 9 - MVC no JSF**




**FONTE: GONÇALVES, 2010**

Descrição das partes do MVC:

- Modelo: O Modelo (*Model*) em uma aplicação é o objeto que representa os dados, manejando e controlando esses dados. O Modelo são as classes que trabalham na busca de dados e no armazenamento, pois não possui nenhum conhecimento em especifico sobre as partes de Controle (*Controller*) e Visão (*View*). Conforme exemplo citado por Gonçalves (2007, p. 386) “um cliente pode ser modelado em uma aplicação, e pode haver vários modos de criar novos clientes ou mudar informações de um relativo cliente”.

- Visão: A Visão (*View*) em uma aplicação é o que manipula a apresentação visual dos dados que foram representados pelo Modelo, apresentando ao usuário os dados resultando do Modelo. Conforme exemplo citado por Gonçalves (2007, p. 386) “uma Apresentação poderá ser um local administrativo onde os administradores se logam em uma aplicação. Cada administrador poderá visualizar uma parte do sistema que outro não vê”.

- Controle: O Controle (*Controller*) em uma aplicação é o objeto que responde as ações que são executadas pelo usuário, atua sobre os dados que o modelo apresentou, decidindo qual Visão deve ser exibida e como o Modelo devera ser revisto ou devera ser alterado. Conforme exemplo citado por Gonçalves (2007, p. 386) “o Controlador recebe um pedido para exibir uma lista de clientes interagindo com o Modelo e entregando uma Apresentação onde esta lista poderá ser exibida”.

O modelo MVC é um padrão de desenvolvimento que auxilia na manutenção do sistema, no desenvolvimento em aplicações Java esse padrão é bem aceito, principalmente se a aplicação é voltada para a Web (GONÇALVES, 2007).

### 2.1.10 Primefaces

Primefaces é uma biblioteca de componentes ricos em Java Server Faces, seus componentes possuem as funcionalidades do Ajax (*Asynchronous Javascript And XML*), baseado na API de Ajax do Java Server Faces. O Primeface possui uma

vasta suíte de componentes, dentre eles podemos citar as tabelas de dados, árvores gráficos, botões, diversos campos de entrada.

Os componentes estão disponíveis no próprio site do fabricante em *showcase* (PRIMEFACES)[3].

### 2.1.11 PostgreSQL

O PostgreSQL é um sistema para gerenciar o banco de dados objeto-relacional de código aberto. O gerenciador PostgreSQL tem uma arquitetura que ganhou comprovadamente uma forte reputação de confiabilidade, integridade de dados e conformidade a padrões, possui mais de 15 anos de desenvolvimento ativo. Roda nos principais sistemas operacionais, como o GNU/Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solares, Tru64), e Microsoft Windows (COMUNIDADE BRASILEIRA DE POSTGRESQL, 2017).

O Banco de dados PostgreSQL oferece algumas funcionalidades, tais como, chaves estrangeiras, visões, gatilhos, integridade transacional, controle de simultaneidade multiversão, entre outros. O usuário tem a possibilidade de estender o PostgreSQL de várias maneiras, adicionando novas funções, tipos de dados, operadores, métodos de índice, funções de agregação e linguagens procedurais (OLIVEIRA, 2006).

O PostgreSQL pode gerenciar um grande volume de dados, alguns limites estão ilustrados na Figura 10.

**Figura 10 - Limite de valores**

| | |
|---|---|
| Tamanho Máximo do Banco de Dados | Ilimitado |
| Tamanho máximo de uma Tabela | 32 TB |
| Tamanho Máximo de uma Linha | 1.6 TB |
| Tamanho Máximo de um Campo | 1 GB |
| Máximo de Linhas por Tabela | Ilimitado |
| Máximo de Colunas por Tabela | 250–1600 dependendo do tipo de coluna |
| Máximo de Índices por Tabela | Ilimitado |

**FONTE: COMUNIDADE BRASILEIRA DE POSTGRESQL, 2017[4]**

[3] Disponível em: <http://www.primefaces.org/showcase/>. Acesso em: 13 de Nov. de 2016.

[4] Disponível em: < http://www.postgresql.org.br/old/sobre>. Acesso em: 21 de Mai. de 2017.

### 2.1.12 Gestão do conhecimento

A gestão do conhecimento se tornou um processo centralizado na organização, devido suas diversas possibilidades de processar, armazenar e consultar informações e dados (ZABOT; SILVA, 2002).

O trabalho em uma organização independente do seu porte é baseado no seu conhecimento, segundo Drucker (1999, p. 40) “somente a organização pode oferecer a continuidade básica de que os trabalhadores do conhecimento precisam para serem eficazes. Apenas a organização pode transformar o conhecimento especializado do trabalho do conhecimento em desempenho”.

A gestão do conhecimento disponibiliza para as organizações a possibilidade de tomar decisões de forma correta, independente do ramo de atividade, pode se tomar decisões referentes a estratégias a serem adotadas sobre os clientes, meios de propaganda de produtos e serviços e até mesmo concorrentes.

### 2.1.13 UML

A UML (Unified Modeling Language) é uma linguagem que pode ser utilizada em domínios de aplicação, softwares que são baseados no paradigma de orientação a objetos são modelados com a linguagem UML.

UML não é uma linguagem de programação, e sim uma linguagem de modelagem, seu objetivo é de auxiliar na definição de características do sistema, dentre eles os seus requisitos, sua estrutura lógica, a dinâmica de seus requisitos, seu comportamento e até mesmo as necessidades físicas com relação ao equipamento sobre o qual a aplicação deverá ser implantada (GUEDES, 2011).

O Diagrama de casos de uso na UML é o diagrama mais geral e informal, normalmente para realizar o levantamento e análise de requisitos do sistema é usado o diagrama de casos de uso. Os casos de uso segundo Guedes (2011, p. 30) “apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter ideia geral de como o sistema irá se comportar”. Identifica os atores (outros sistemas, usuários) que utilizarão o sistema, e os serviços, que são as funcionalidades que o software disponibilizará aos atores.

O diagrama de classes é um dos mais importantes da UML e provavelmente o mais utilizado. A estrutura de classes utilizadas pelo sistema é definida pelo diagrama de classes, determinando os métodos e atributos que cada classe tem, determina o relacionamento entre as classes e como ocorre a troca de informação entre si (GUEDES, 2011).

# 3 METODOLOGIA

Em uma aplicação web, para realizar o seu desenvolvimento é necessário utilizar tecnologias específicas.

Para realizar o armazenamento das informações (persistência), será utilizado JPA em conjunto com o Hibernate e o banco de dados PostgreSQL, que será instalado e configurado em um servidor específico para o sistema proposto, juntamente com o PostgreSQL vai estar o glassfish, que possibilitará que a aplicação Java Web utilizando o framework JSF seja executada.

A aplicação será desenvolvida para usuários que irão utilizar a Web, o sistema vai ser desenvolvido com a biblioteca Primefaces, que irá prover a interface para o usuário.

O código-fonte vai ser escrito na linguagem Java, toda a requisição que o usuário solicitar será tratado pela JSF, mais específico na camada de controle.

## 3.1 ESTUDO DO SISTEMA EXISTENTE

O Processo de atendimento do suporte técnico da empresa Software PF atualmente é realizado da seguinte forma, inicia com a solicitação de suporte, momento em que o cliente realiza contato solicitando suporte através do atendimento online no site da empresa Software PF, por telefone ou e-mail. Após acontece a solicitação de informações, é quando o analista de suporte técnico realiza a coleta de informações necessárias para conduzir o suporte, dentre elas estão o problema que está ocorrendo. Em seguida o técnico realiza a consulta dos dados do cliente em um sistema de gestão da empresa Software PF.

Para consultar o problema o analista realiza uma verificação no Dropbox, pesquisando os documentos para tentar identificar uma possível resolução do problema, caso tenha encontrado, o analista aplica a correção, caso contrário é necessário realizar os procedimentos na tentativa e erro sem ter um processo para auxiliar. Depois de resolver o problema do cliente e finalizar o atendimento, se for um novo processo de resolução, o analista encaminha um e-mail para o responsável por armazenar os documentos no Dropbox.

Para a validação de resolução, o responsável que recebeu o e-mail efetua os devidos testes validando todas as informações. Se o processo estiver correto, o mesmo é armazenado no Dropbox.

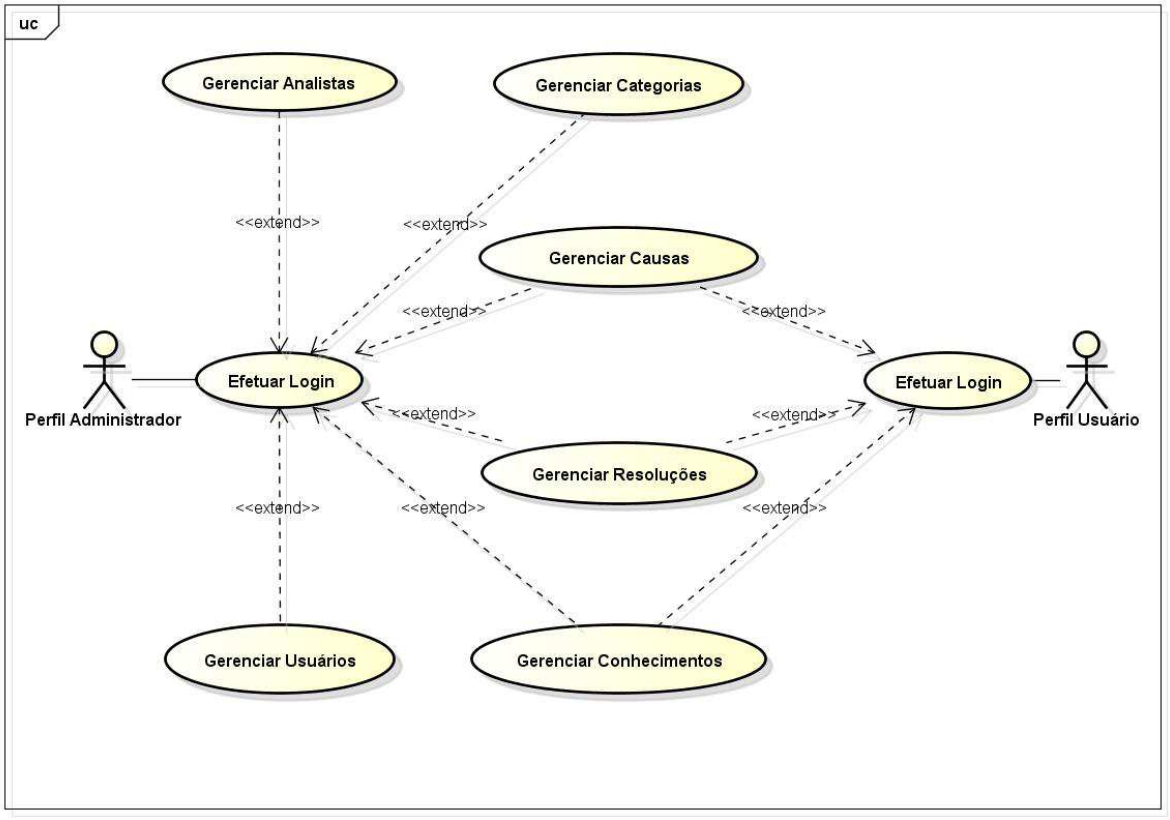
# 4 MODELAGEM DO SISTEMA

Com base no estudo do sistema existente a no levantamento de requisitos, foi elaborado a modelagem do sistema proposto. O sistema de gestão do conhecimento tem o intuito de agilizar e padronizar os atendimentos no suporte técnico.

## 4.1 DIAGRAMA DE CASOS DE USO DA GESTÃO DO CONHECIMENTO

O sistema será desenvolvido com dois perfis, que serão os níveis de segurança das informações contidas na aplicação, cada usuário irá pertencer a um perfil específico, e nesse perfil irá conter as permissões de utilização do sistema conforme ilustrado na Figura 11.

**Figura 11 - Diagrama de Casos de Uso**




**FONTE: DO AUTOR**

De forma descritiva e hierárquica crescente em relação à figura 11, o usuário com perfil de administrador terá os privilégios de administrador, que são todas as ações que o sistema poderá realizar, o perfil administrador terá a possibilidade de gerenciar analistas, usuários, categorias, causas, resoluções e conhecimentos.

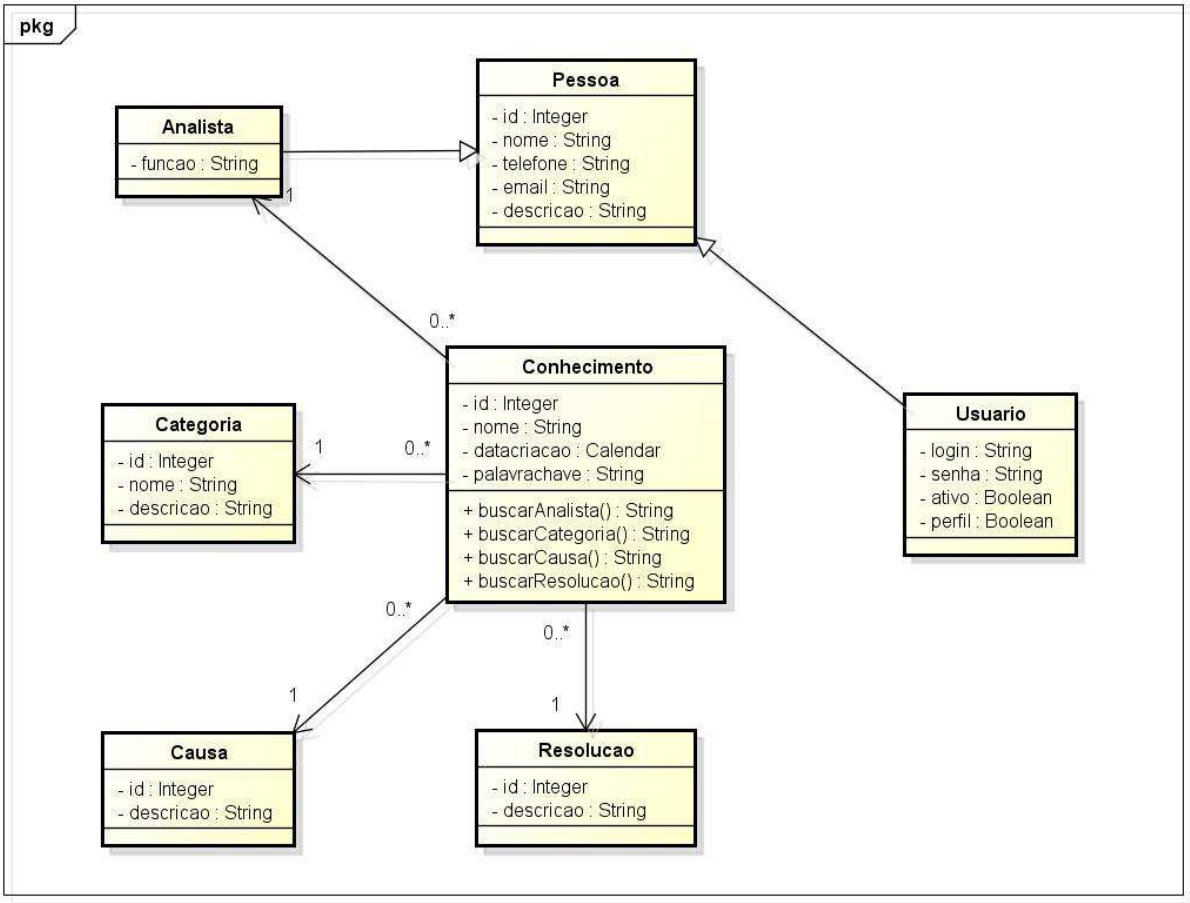
O usuário com perfil de usuário poderá apenas gerenciar causas, resoluções e conhecimentos, essa delimitação se dá ao perfil atribuído, que poderá somente realizar essa tarefa no sistema.

Para realizar qualquer ação no sistema é necessário estar logado, ao acessar o sistema, a primeira informação solicitada pela aplicação será o usuário e senha, caso o usuário não apresentar essas informações, o mesmo terá que fazer uma requisição para o usuário de perfil administração.

## 4.2 DIAGRAMA DE CLASSES DA GESTÃO DO CONHECIMENTO

O Diagrama de classes para o sistema de gestão do conhecimento está ilustrado na figura 12.

**Figura 12 - Diagrama de Classes**




**FONTE: DO AUTOR**

Neste modelo foram utilizadas as seguintes classes:

- **Usuario**

  A classe Usuario tem por objetivo armazenar todos os usuários do sistema, sem limite de armazenamento de usuários. Está associada com a classe Pessoa

  A classe Usuario possui os atributos login do usuário (login) do tipo String, senha do usuário (senha) do tipo String, ativo (ativo) do tipo Boolean, e perfil (perfil) do tipo Boolean

**Pessoa**

A classe Pessoa tem por objetivo armazenar as pessoas que utilizarão o sistema, não possui limite de armazenamento de pessoas. Está associada a duas classes, representada pela classe Usuario e Analista.
A classe Pessoa possui os atributos id da pessoa (id) do tipo int, nome da pessoa (nome) do tipo String, telefone da pessoa (telefone) do tipo String, e-mail da pessoa (email) do tipo String e descrição da pessoa do tipo String.

- **Analista**

  A classe Analista tem por objetivo armazenar os analistas, que são os técnicos responsáveis pelos atendimentos, não possui limite de armazenamento de analistas. Está associada a duas classes, representada pela classe Conhecimento e Pessoa.
  A classe Analista possui apenas o atributo função do analista (funcao) do tipo String.

- **Resolucao**

  A classe Resolucao tem por objetivo armazenar as resoluções do sistema, que são as informações necessárias para resolver o problema, não possui limite de armazenamento de resoluções. Está associada a uma classe, representada pela classe Conhecimento.
  A classe Resolucao possui os atributos id da resolução (id) do tipo int e descrição da resolução (descricao) do tipo String.

- **Causa**

  A classe Causa tem por objetivo armazenar todas as causas que originou os problemas, não possui limite de armazenamento de causas. Está associada a uma classe, representada pela classe Conhecimento.
  A classe Causa possui os atributos id da causa (id) do tipo int e descrição da causa (descricao) do tipo String.

- **Categoria**

  A classe Categoria tem por objetivo armazenar todos as categorias que são vinculadas aos equipamentos e softwares que são comercializados pela empresa Software PF, não possui limite de armazenamento de categorias. Está associada a uma classe, representada pela classe Conhecimento.

  A classe Categoria possui os atributos id da categoria (id) do tipo int, nome da categoria (nome) do tipo String e descrição da categoria (descricao) do tipo String.

- **Conhecimento**

  A classe Conhecimento tem por objetivo armazenar todos os conhecimentos, que são os atendimentos que foram solucionados pelos analistas, não possui limite de armazenamento de conhecimentos. Está associada a quatro classes, representada pela classe Resolucao, Analista, Categoria e Causa.

  A classe Conhecimento possui os atributos id do conhecimento (id) do tipo int, nome do conhecimento (nome) do tipo String, data da criação do conhecimento (datacriacao) do tipo Calendar e palavra chave do conhecimento (palavrachave) do tipo String.

# 5 DESENVOLVIMENTO

Nesse tópico será abordado toda a estrutura e processos realizados para desenvolver o sistema de gestão web.

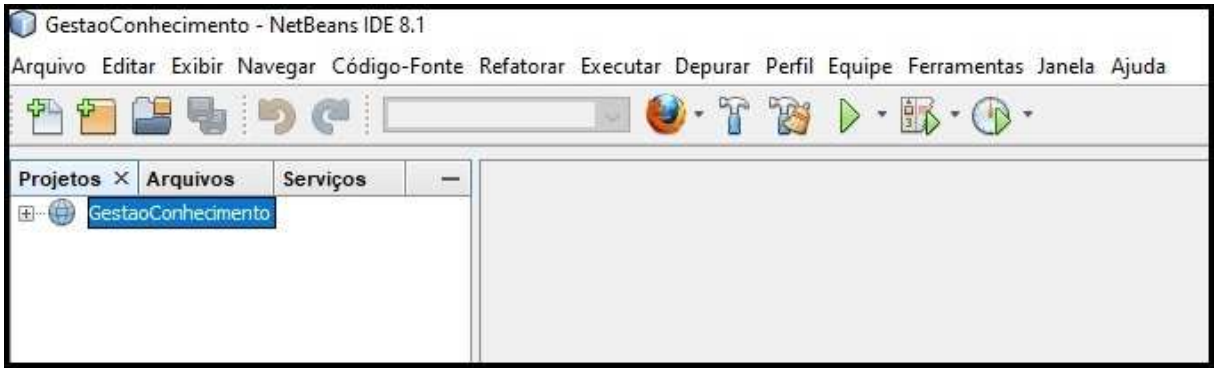
## 5.1 AMBIENTE DE DESENVOLVIMENTO

Antes de iniciar o desenvolvimento do sistema, foi realizado um estudo das tecnologias utilizadas e suas respectivas versões, com o propósito de evitar conflitos ou perdas de funcionalidades.

### 5.1.1 IDE

A IDE utilizada para o desenvolvimento do sistema foi o NetBeans versão 8.1 ilustrado da Figura 13, IDE que possibilita o desenvolvimento de aplicações em Java que prove suporte ao JPA, JSF, Hibernate, Glassfish, PostgreSQL e Primefaces.

**Figura 13 - IDE NetBeans versão 8.1**

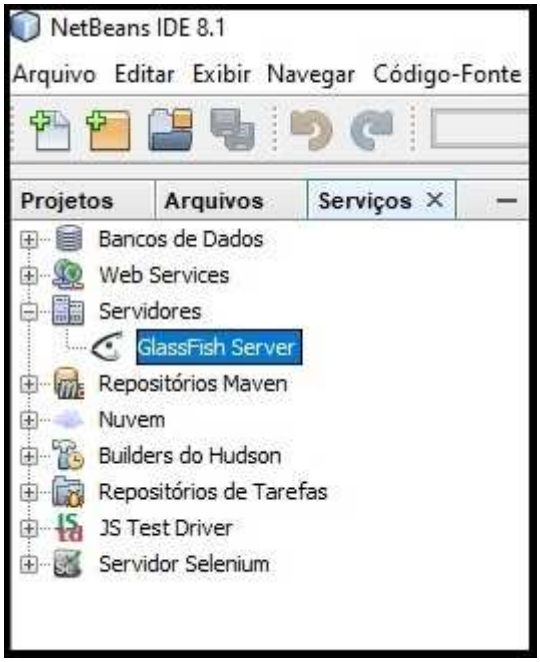



**FONTE: DO AUTOR**

### 5.1.2 Servidor e SGBD

É necessário possuir um servidor para que o sistema seja devidamente executado, o servidor utilizado foi o GlassFish Server versão 4 (Figura 14).

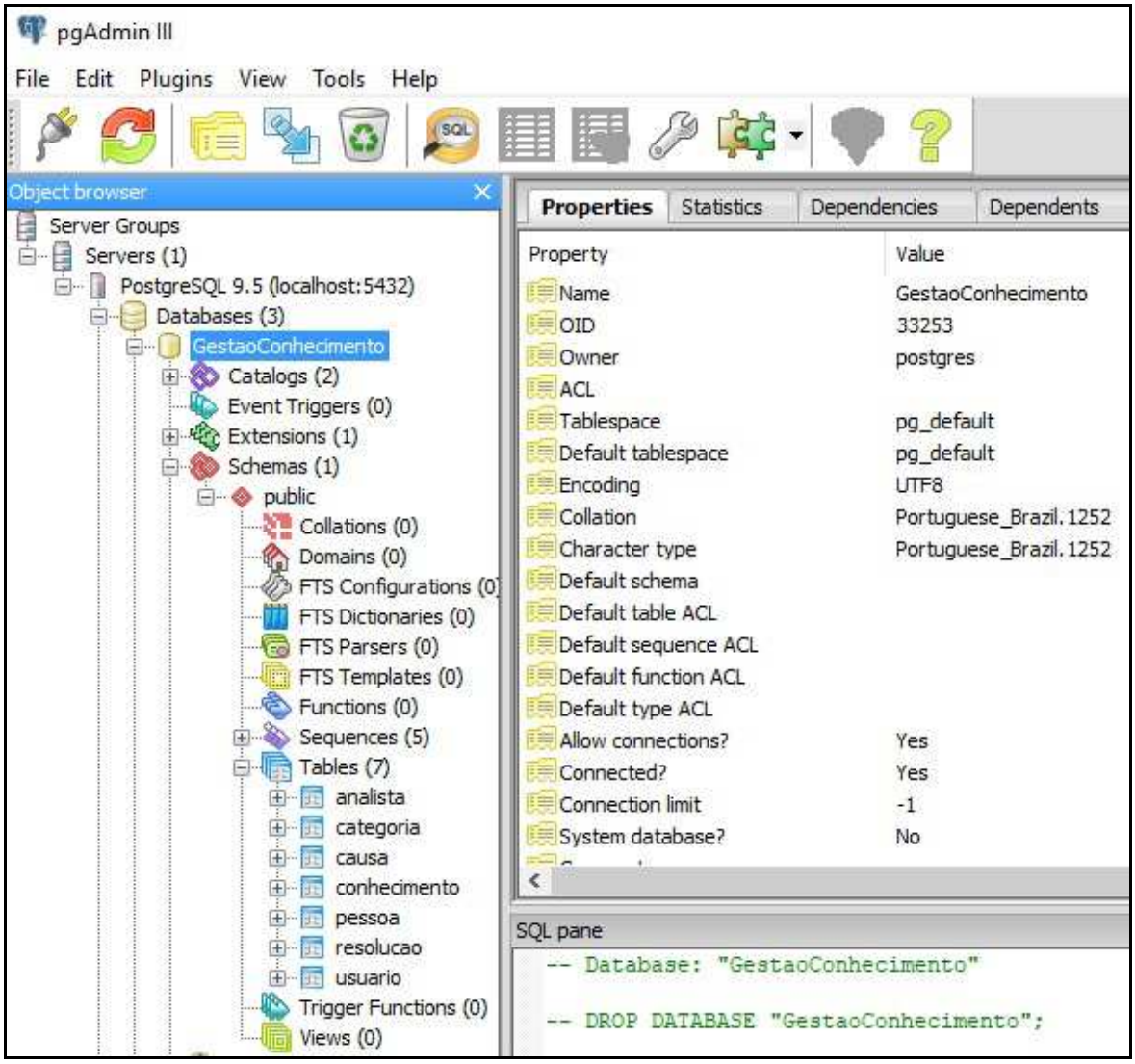
**Figura 14 - Servidor GlassFish versão 4**




**FONTE: DO AUTOR**

Utilizou-se o banco de dados PostgreSQL versão 9.5 juntamente com o sistema de gerenciamento de banco de dados PGAdmin III, ambos ilustrados na figura 15, imagem que também podemos verificar toda a estrutura do banco “GestaoConhecimento” que mantem armazenado todas as informações do sistema desenvolvido.

**Figura 15 - PostgreSQL e PGAdmin III**




**FONTE: DO AUTOR**

### 5.1.3 Estrutura do sistema

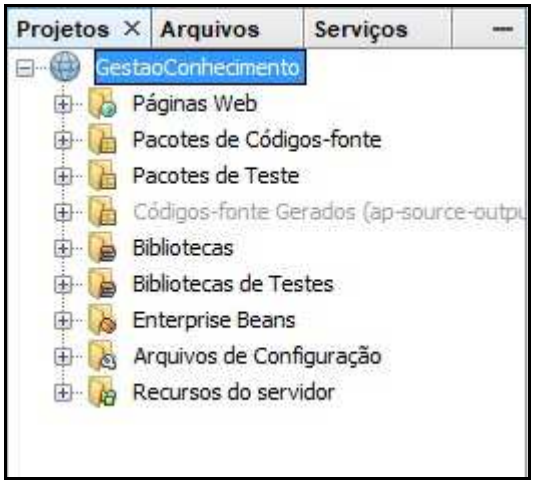
O sistema foi desenvolvido utilizando o MVC, ambos foram modelados e armazenados na mesma estrutura do sistema, sem a necessidade de criar outros projetos para efetuar a importação das classes e modelos.

Em páginas Web, estão armazenados a camada de visão do projeto, que são as páginas e layout onde o usuário irá realizar a interação com o sistema.

Nos pacotes de Código-fonte estão armazenados o restante do MVC, o controle e modelo. Possui também os conversores, as mensagens de validação e as classes de testes que foram utilizadas durante o desenvolvimento do projeto.

As bibliotecas, os arquivos de configurações, e recursos do servidor ficam em pastas separadas, porém na mesma hierarquia das páginas Web e pacotes de código fonte, mais detalhes na figura 16.

**Figura 16 - Estrutura do Sistema**




**FONTE: DO AUTOR**

### 5.1.4 Bibliotecas

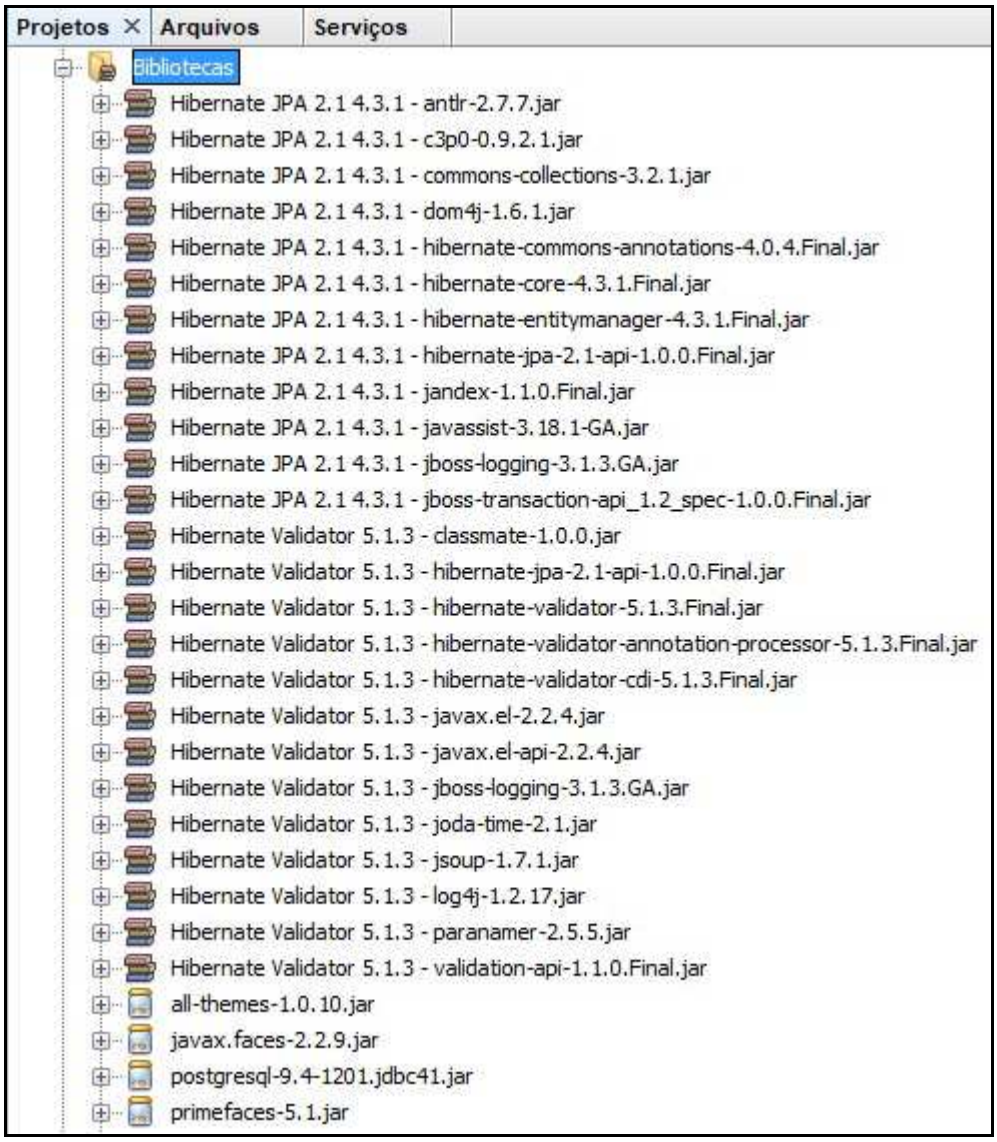
Os recursos utilizados para elaborar o sistema de gestão do conhecimento foram providos de bibliotecas, tais recursos foram utilizados para validar as ações do usuário, persistir os dados no banco de dados e a layout da interface com o usuário (Figura 17).

As bibliotecas incluídas no sistema são:

- Biblioteca Hibernate JPA 2.1 4.3.1: Biblioteca Hibernate JPA foi utilizada para efetuar o mapeamento relacional dos objetos e prover a persistência dos dados no banco de dados.
- Biblioteca Hibernate Validator 5.1.3: Biblioteca Hibernate Validator foi utilizada para efetuar a validação dos dados que são persistidos no banco de dados.
- JDBC PostgreSQL versão 9.4.1201: Driver JDBC do PostgreSQL disponibiliza recursos para realizar a conexão entre o sistema e o banco de dados e manipulação dos dados das tabelas.

- Biblioteca PrimeFaces 5.1: Biblioteca PrimeFaces foi utilizada para criar os recursos gráficos de interface, auxiliando na organização e melhoria do layout.

**Figura 17 - Bibliotecas utilizadas**




**FONTE: DO AUTOR**

### 5.1.5 Camada modelo

Os objetos no sistema são instanciados pela camada de modelo, as classes da camada modelo contém as anotações necessárias para que possa ser realizado o mapeamento do objeto relacional para integração com o banco de dados.

Assim como as demais classes que serão identificadas como entidade, foi inserido a anotação @Entity, e para declarar um nome especifico na tabela do banco de dados foi utilizada a anotação @Table, a Figura 18 ilustra um trecho do código da classe Conhecimento com suas anotações e atributos.

**Figura 18 - Trecho do código da classe Conhecimento**

```
@Entity
@Table(name = "conhecimento")
public class Conhecimento implements Serializable {

    @Id
    @Column(name = "id")
    @SequenceGenerator(name = "seq_id_conhecimento", sequenceName = "gen_conhecimento_id",
            allocationSize = 1)
    @GeneratedValue(generator = "seq_id_conhecimento", strategy = GenerationType.SEQUENCE)
    private Integer id;

    @Length(min = 2, max = 50, message = "O nome deve ter entre {min} e {max} caracteres")
    @NotEmpty(message = "O nome deve ser informado")
    @Column(name = "nome", length = 50, nullable = false, unique = true)
    private String nome;

    @NotNull(message = "A data deve ser informada")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "datacriacao", nullable = false)
    private Calendar datacriacao;

    @Length(min = 2, max = 200, message = "A Palavra chave deve ter entre {min} e {max} caracteres")
    @NotEmpty(message = "A Palavra chave deve ser informado")
    @Column(name = "palavrachave", length = 200, nullable = false)
    private String palavrachave;

    private byte[] anexo;

    @NotNull(message = "A Resolução deve ser informada")
    @ManyToOne
    @JoinColumn(name = "resolucao", referencedColumnName = "id", nullable = false)
    private Resolucao resolucao;
```

**FONTE: DO AUTOR**

Conforme ilustrado na Figura 18, tem-se algumas anotações que são a @Column, utilizada para informar o nome da coluna em uma tabela no banco de dados. O @id é utilizado para definir o identificador da tabela. A geração utilizada foi a sequencial, a mesma é definida pela anotação @GenerationType.SEQUENCE, o valor da sequência é gerado e salvo no banco de forma automática, apenas informando a anotação @GenerateValue, o gerador utiliza a anotação @SequenceGenerator.

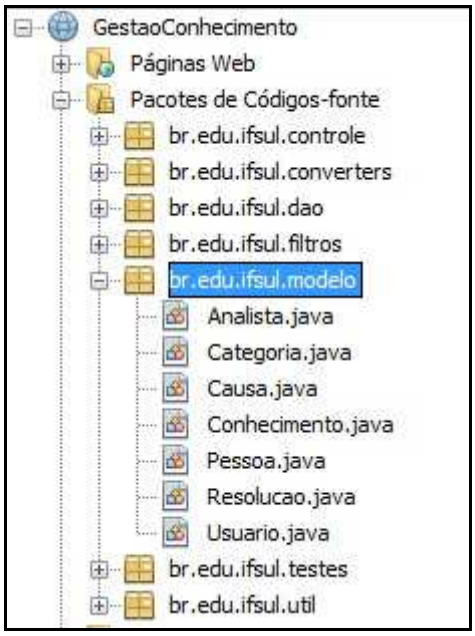
Os atributos são validados por anotações que recebem os valores da interface, na classe Conhecimento foram utilizadas as anotações, @Length, que controla o valor mínimo e valor máximo que o atributo pode receber. A anotação @NotEmpty verifica se o valor que o atributo está recebendo é vazio. Anotação

@NotNull verifica os valores do atributo e bloqueia caso receber um valor nulo. Para determinar que o atributo receba valores referente a tempo se usa a anotação @Temporal, tal anotação é utilizada por exemplo, sistema necessita de um atributo de data.

Para os relacionamentos entre as entidades são necessárias anotações especificas, por exemplo, @JoinColumn, a mesma consiste em possuir o nome da coluna que foi referenciada pela chave estrangeira.

O sistema de gestão do conhecimento possui as seguintes classes na camada de modelo: Analista.java, Categoria.java, Causa.java, Conhecimento.java, Pessoa.java, Resolucao.java, usuário.java, ambas ilustradas na figura 19.

**Figura 19 - Estrutura da Camada Modelo**




**FONTE: DO AUTOR**

A pasta Arquivos de Configuração contém o arquivo persistence.xml, o mesmo possui as informações para que a JPA possa realizar a comunicação com o banco de dados. A Figura 20 ilustra um treco do código do arquivo.

**Figura 20 - Trecho do código do arquivo persistence.xml**

```
<persistence-unit name="GestaoConhecimentoPULocal" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
        <property name="javax.persistence.jdbc.url"
                  value="jdbc:postgresql://localhost:5432/GestaoConhecimento"/>
        <property name="javax.persistence.jdbc.password" value="postgres"/>
        <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
        <property name="javax.persistence.jdbc.user" value="postgres"/>
        <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
        <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
        <property name="hibernate.connection.autocommit" value="false"/>
        <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
</persistence-unit>
```

**FONTE: DO AUTOR**

O nome da unidade de persistência é definido no elemento <persistence-unit> como GestaoConhecimentoPUlocal, utilizado para identificar a unidade de persistência no sistema. O provedor de persistência foi definido como HibernatePersistence, o mesmo é informado no elemento <provider>.

Para definir as propriedades de conexão com o banco de dados é usado o conteúdo <property name=”javax.persistence.jdbc.url”>, o driverJDBC do PostgreSQL é definido como <property name=”javax.persistence.jdbc.driver”>, o dialeto padrão do SQL utilizado pelo Hibernate é setado na propriedade <property name=”hibernate.dialect”>. As demais informações de conexão com o banco de dados são de preenchimento obrigatório, tais como local, porta, nome do banco de dados, usuário e senha.

### 5.1.5.1 Classes DAO

O pacote br.edu.ifsul.dao contém as classes DAO, que são responsáveis pela comunicação entre a unidade de persistência, os elementos do sistema e o banco de dados. Foi elaborada uma classe DAO especifica para o projeto, chamada de DAOgenerico, a mesma realiza funções como, quantidade de registros que são exibidos por página, paginação caso contenha um número elevado de registros, filtros e ordenação das pesquisas realizadas. Dessa forma as demais classes DAO possuem códigos mais enxutos, conforme figura 21, podemos ver um trecho do código da classe DAOGenerico.

**Figura 21 - Trecho do Código da classe DAOGenerico**

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

    @PersistenceContext(unitName = "GestaoConhecimentoPU")
    private EntityManager em;
    private Class classePersistente;
    private String ordem = "id";
    private String filtro = "";
    private Integer maximoObjetos = 5;
    private Integer posicaoAtual = 0;
    private Integer totalObjetos = 0;
```

**FONTE: DO AUTOR**

As demais classes DAO, recebem funções para implementar os métodos da classe DAOGenerico, com intuito de facilitar a codificação, pois é utilizado reaproveitamento de código. A figura 22 ilustra o código principal da classe ConhecimentoDAO.

**Figura 22 - Código da classe ConhecimentoDAO**

```
@Stateless
public class ConhecimentoDAO<T> extends DAOGenerico<Conhecimento> implements Serializable {

    public ConhecimentoDAO(){
        super();
        super.setClassePersistente(Conhecimento.class);
    }

}
```

**FONTE: DO AUTOR**

### 5.1.6 Camada de controle

A camada de controle é utilizada para realizar a comunicação com a interface e os elementos da camada de persistência. No sistema de gestão do conhecimento, foram definidos os métodos para exibir a página requisitada, salvar, editar e remover os objetos (figura 23).

**Figura 23 - Código da classe ControleConhecimento**

```
public String listar() {
    return "/privado/conhecimento/listar?faces-redirect=true";
}

public void salvar() {
    try {
        if (objeto.getId() == null) {
            dao.persist(objeto);
        } else {
            dao.merge(objeto);
        }
        UtilMensagens.mensagemInformacao("Conhecimento salvo com sucesso!");
    } catch (Exception e) {
        UtilMensagens.mensagemErro("Erro ao salvar conhecimento: " + e.getMessage());
    }
}

public void editar(Integer id) {
    try {
        objeto = dao.getObjectById(id);
    } catch (Exception e) {
        UtilMensagens.mensagemErro("Erro ao buscar conhecimento: " + e.getMessage());
    }
}

public void remover(Integer id) {
    try {
        objeto = dao.getObjectById(id);
        dao.remove(objeto);
        UtilMensagens.mensagemInformacao("Conhecimento removido com sucesso!");
    } catch (Exception e) {
        UtilMensagens.mensagemErro("Erro ao remover conhecimento: " + e.getMessage());
    }
```

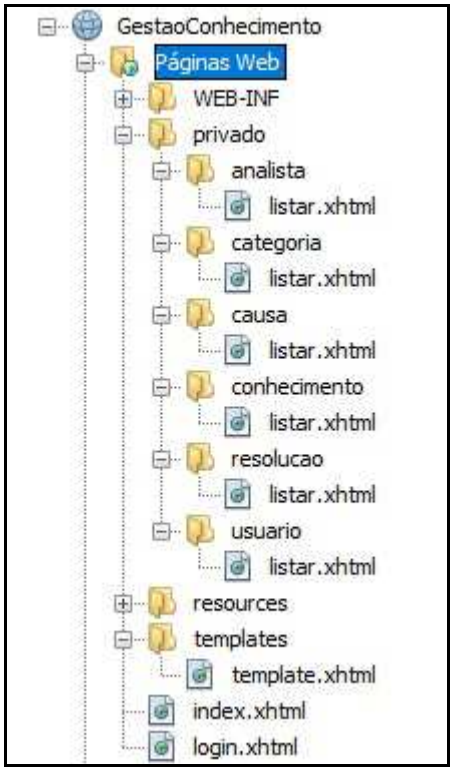
**FONTE: DO AUTOR**

Quando o método listar é chamado, o mesmo realiza o redirecionamento e envia a solicitação para a página requisitada, nesse caso a solicitação é para a página listar do conhecimento. O método salvar busca o id do objeto e verifica o valor do atributo, se for nulo o método salva um novo objeto no banco de dados, caso contrário é realizado a alteração dos dados e salvo o mesmo objeto. O método editar realiza a busca dos dados salvos no banco de dados, a busca é feita pelo id do objeto. O método remover realiza a busca do objeto e efetua a remoção dos dados no banco de dados, semelhante ao método editar, a sua busca também é feita pelo id do objeto. Todas as mensagens de ações são tratadas com o *UtilMensagens.mensagem*Informacao e *UtilMensagens.mensagem*Erro.

### 5.1.7 Camada de visão

As páginas web do sistema de gestão do conhecimento foram desenvolvidas utilizando o framework JSF e a biblioteca PrimeFaces. Dentro da camada de visão possui a pasta templates, que contém o arquivo template.xhtml, esse arquivo é responsável pela exibição do conteúdo da interface para o usuário, possui tags que são inseridas nas páginas web do sistema, a figura 24 ilustra a estrutura da camada de visão.

**Figura 24 - Estrutura da Camada de Visão**




**FONTE: DO AUTOR**

O arquivo template.xhtml possui a tag <p:menu>, responsável pela exibição do menu na página onde o arquivo template.xhtml for chamado, a tag <p:submenu> é responsável por exibir o submenu da página, nessa mesma tag foi inserido um controle de exibição dos conteúdos, onde apenas usuários logados e ativos conseguem ter acesso ao submenu. A tag <p:menuitem> é responsável por exibir cada item do menu, ambas as tags pertencem a biblioteca PrimeFaces. O código do menu do sistema pode ser visualizado na figura 25.

**Figura 25 – Código do menu do sistema**

```
<p:menu>
    <p:submenu label="Cadastros" rendered="#{(controleLogin.usuarioLogado != null)
                                             and (controleLogin.usuarioLogado.ativo == true)
                                             and (controleLogin.usuarioLogado.perfil == true)}">

        <p:menuitem value="Inicio" icon="ui-icon-home"
                    action="#{controlePrincipal.home()}" ajax="false"/>

        <p:menuitem value="Categoria" icon="ui-icon-note"
                    action="#{controleCategoria.listar()}" ajax="false"/>

        <p:menuitem value="Analista" icon="ui-icon-person"
                    action="#{controleAnalista.listar()}" ajax="false"/>

        <p:menuitem value="Causa" icon="ui-icon-note"
                    action="#{controleCausa.listar()}" ajax="false"/>

        <p:menuitem value="Resolução" icon="ui-icon-note"
                    action="#{controleResolucao.listar()}" ajax="false"/>

        <p:menuitem value="Usuário" icon="ui-icon-person"
                    action="#{controleUsuario.listar()}" ajax="false"/>

        <p:menuitem value="Conhecimento" icon="ui-icon-note"
                    action="#{controleConhecimento.listar()}" ajax="false"/>
    </p:submenu>
```

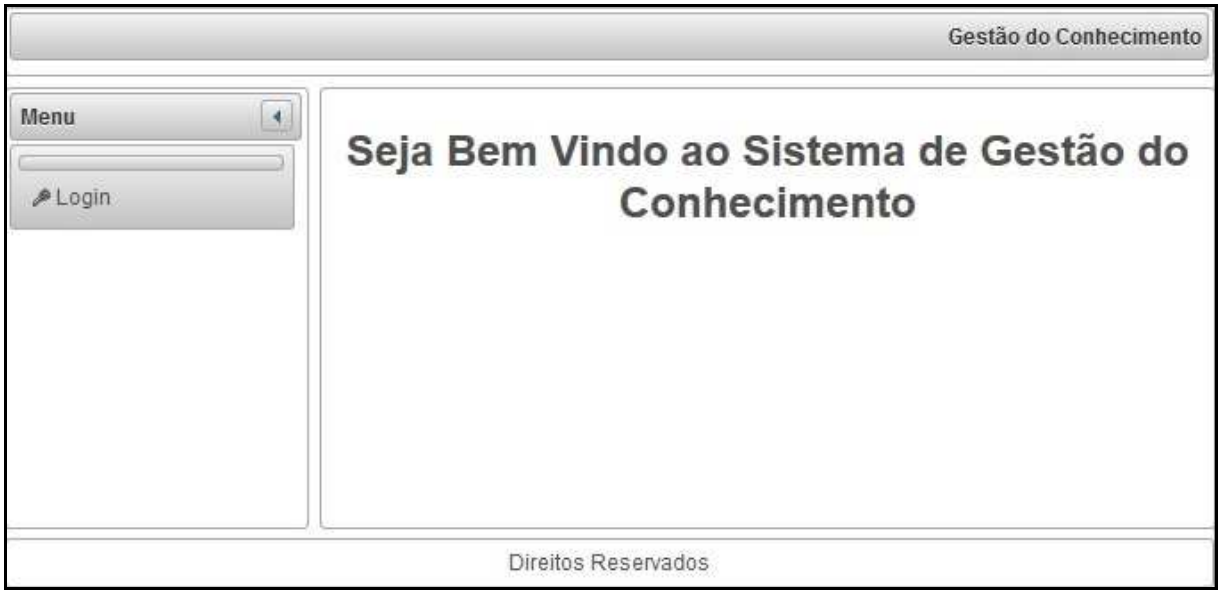
**FONTE: DO AUTOR**

# 6 RESULTADOS

O sistema de gestão do conhecimento foi desenvolvido com o objetivo de agilizar e padronizar a resolução dos problemas, os conhecimentos que são cadastrados possuem suas categorias, causas dos problemas e suas resoluções.

Neste tópico será demostrado a utilização do sistema. A Figura 26 demonstra a tela inicial do sistema, para ter acesso é necessário selecionar o Login no Menu localizado a esquerda.

**Figura 26 - Pagina Inicial do Sistema de Gestão do Conhecimento**




**FONTE: DO AUTOR**

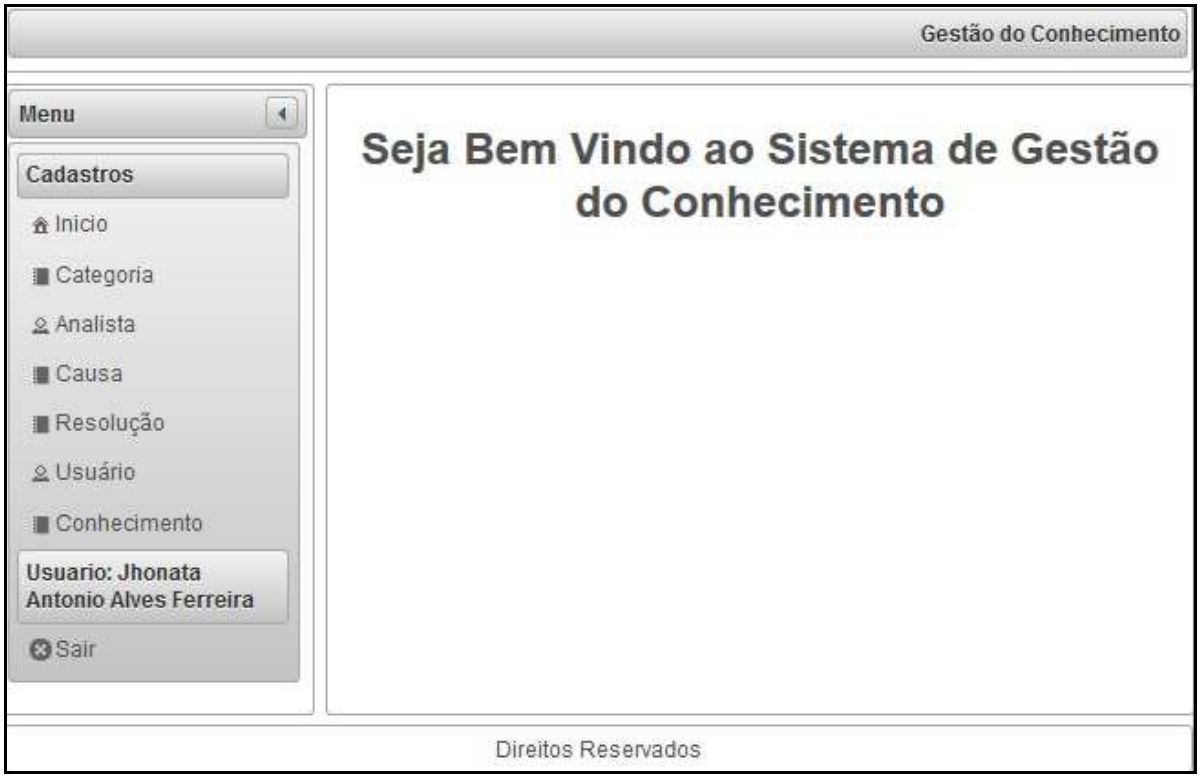
O sistema possui validações de acesso, somente usuários ativos conseguem visualizar o menu, ao informar o login e senha (Figura 27) é realizado uma verificação no perfil do usuário, se o mesmo for administrador será liberado o menu completo do sistema, caso contrário o menu irá liberar apenas as opções de gerenciamento de causas, resoluções e conhecimentos.

**Figura 27 - Validação de acesso ao Sistema**


**FONTE: DO AUTOR**

Após acessar o sistema com perfil de administrador será apresentado o menu completo do sistema, independente da página que for apresentada, o nome do usuário logado será sempre exibido no menu (Figura 28).

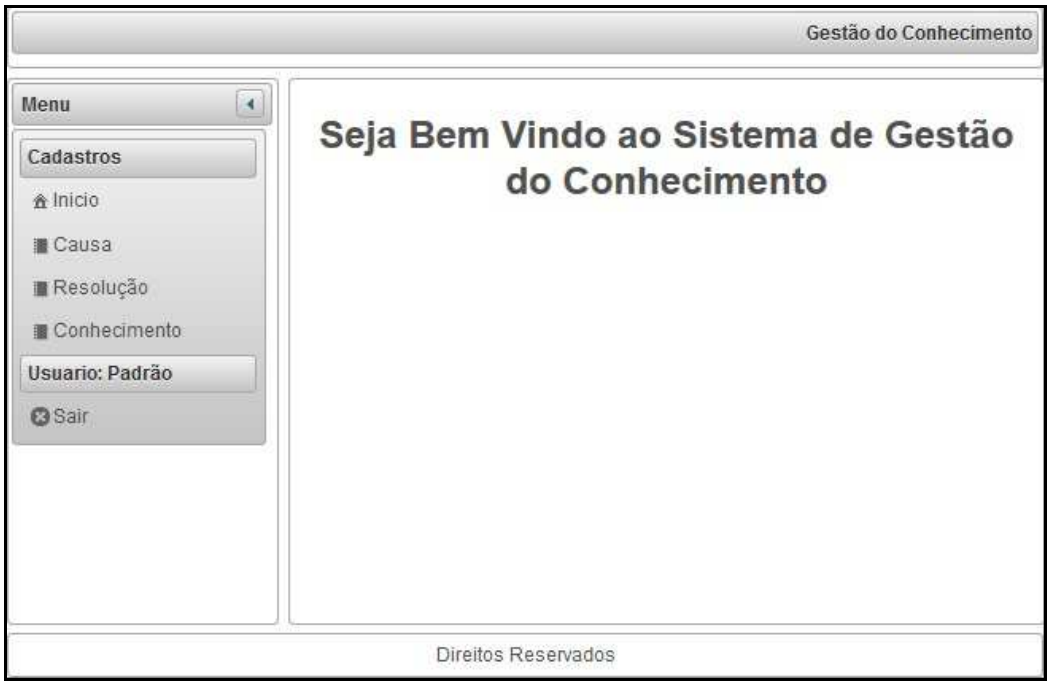
**Figura 28 - Pagina inicial do perfil administrador**




**FONTE: DO AUTOR**

Se o usuário informado não possuir perfil de administrador o sistema será apresentado com o menu restrito, o nome do usuário logado será sempre exibido no menu. A figura 29 ilustra o menu restrito.

**Figura 29 - Página inicial perfil restrito**




**FONTE: DO AUTOR**

As funcionalidades de cadastro, edição e remoção dos dados foram aplicadas para todas as páginas do sistema, como também a parte de ordenação e filtro dos dados, será usado o gerenciamento dos conhecimentos como demonstrativo. A figura 30 ilustra a listagem dos registros da página inicial dos conhecimentos.

**Figura 30 - Listagem dos Conhecimentos**


**FONTE: DO AUTOR**

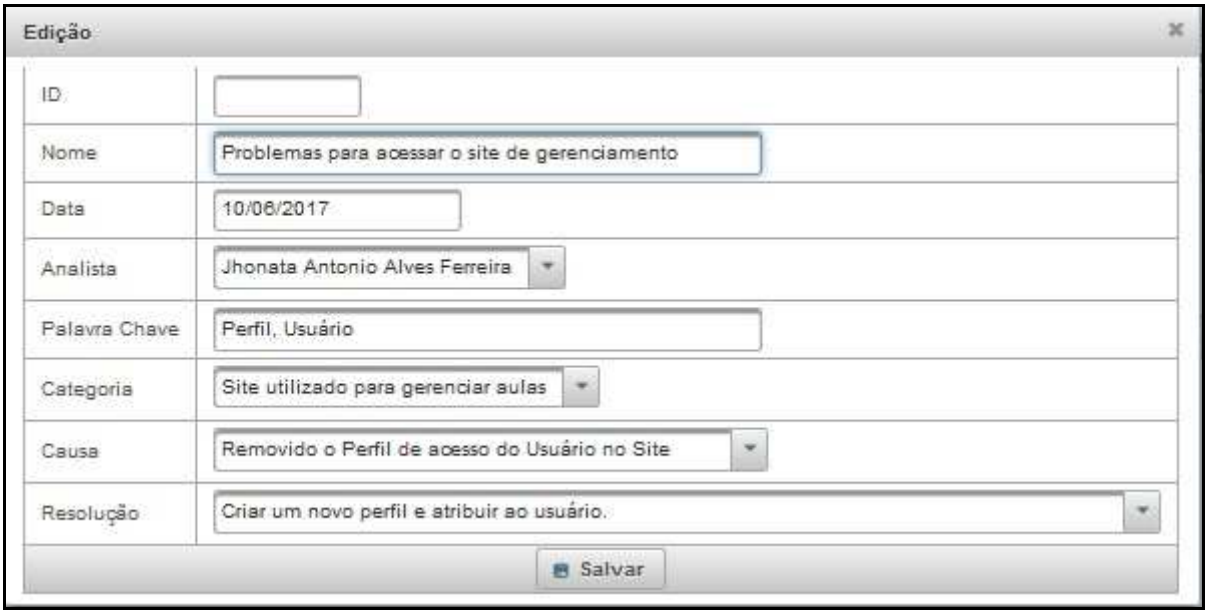
**Os campos Ordem e Filtro** ambos configuram a listagem dos registros, poderá ser selecionado a ordem desejada e no campo filtro digitar o que deseja pesquisar; O campo **Máximo de objetos** está configurado como padrão para buscar 5 registros, caso necessite alterar o valor, apenas deve-se informar o número de registros desejados; O botão **Primeiro** é utilizado caso queira voltar a primeira página da listagem dos conhecimentos; O botão **Último** redireciona para a última página da listagem dos conhecimentos. Os botões **Anterior e Próximo** são utilizados para voltar ou avançar as páginas. Na figura 31 será demonstrado uma pesquisa selecionando a ordem palavra chave e digitando GPS no campo filtro, a busca será feita de forma automática.

**Figura 31 - Pesquisa de Conhecimentos**


**FONTE: DO AUTOR**

Um novo conhecimento pode ser inserido ao clicar no botão **Novo Conhecimento**, será aberto o formulário para preenchimento das informações, a figura 32 ilustra a criação de um novo conhecimento e após preencher os campos é necessário clicar no botão salvar, o formulário irá fechar e o conhecimento aparecerá automaticamente na listagem dos registros.

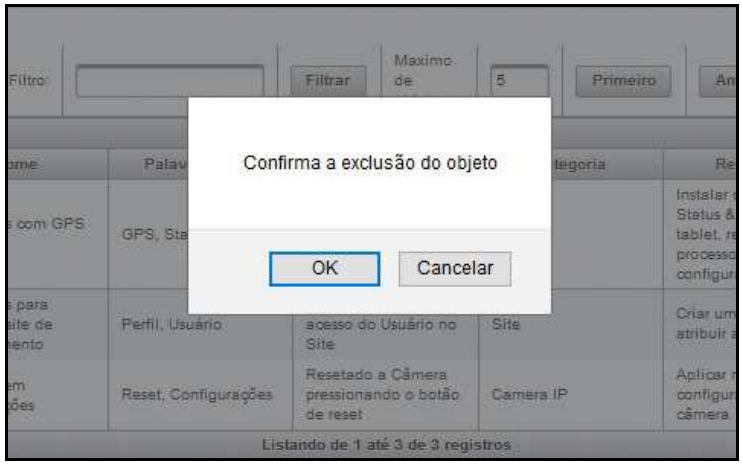
**Figura 32 - Cadastrando um novo conhecimento**




**FONTE: DO AUTOR**

Para editar um conhecimento já cadastrado no sistema, é preciso acionar o **botão editar** que está representado com a imagem de um **lápis**, caso seja necessário remover um conhecimento, é preciso acionar o **botão excluir** que está representado com a imagem de uma **lixeira**. Por motivos de segurança, quando o botão excluir for acionado, sempre será exibido uma caixa de confirmação de exclusão (Figura 33).

**Figura 33 - Removendo um conhecimento**




**FONTE: DO AUTOR**

Após o uso do sistema de gestão do conhecimento, o usuário tem a possibilidade de sair do sistema acionando **o botão Sair**. Figura 34 ilustra o botão Sair que está localizado no Menu.

**Figura 34 - Saindo do Sistema**


**FONTE: DO AUTOR**

# 7 CONSIDERAÇÕES FINAIS

Levando em consideração a concorrência atual do mercado, a elaboração de uma aplicação, com objetivo de agilizar e padronizar a resolução dos problemas em um ambiente de suporte técnico, pode ser considerado um ponto importante, pois qualidade e agilidade é um diferencial que a empresa pode oferecer para seus clientes.

Foi necessário realizar um estudo sobre o funcionamento dos processos de um suporte técnico, com base nessas informações o sistema foi desenvolvido de modo genérico, o mesmo poderá ser utilizado para qualquer segmento de suporte técnico, pois utiliza nomenclaturas que se adaptam facilmente.

Os estudos relacionados ao Java foram satisfatórios, os códigos ficam mais organizados, as bibliotecas utilizadas para elaborar o layout e validar os dados de inserção facilitou o desenvolvimento do sistema.

A versão atual do sistema está operando conforme foi realizado o levantamento dos requisitos, os testes locais foram realizados juntamente com o supervisor do setor de suporte técnico da empresa SoftwarePF, concluísse que com o novo sistema a forma de atendimento será mais ágil e a resolução dos problemas terá um padrão de solução, por motivos de segurança e processos internos da SoftwarePF, até o momento não foi possível realizar a implantação na empresa.

Após os testes houve a necessidade de o sistema gerar relatórios dos conhecimentos e anexar imagens ao cadastrar um novo conhecimento, tais funcionalidades serão implementadas em uma versão futura.

**REFERÊNCIAS**

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

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

DEITEL, Harvey; DEITEL, Paul. Java como Programar. 8. Ed. São Paulo. Prentice Hall, 2010.

DRUCKER, Peter. Desafios Gerenciais para o Século XXI. São Paulo. Pioneira, 1999.

GEARY, David; HORSTMANN, Cay. Core JavaServer Faces Fundamentos. 2. Ed. Rio de Janeiro. Alta Books, 2007.

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

GONÇALVES, Edson. Desenvolvendo Aplicações Web com JSP, servlets, JavaServer Faces, Hibernate, EJB 3 Persistence e Ajax. Rio de Janeiro. Ciência Moderna, 2007.

GONÇALVES, Edson. JavaServer Faces 2.0 na Prática – Parte 3. Gonçalves. 2010. Disponível em: <http://www.edsongoncalves.com.br/2010/02/21/javaserver-faces-2-0-na-pratica-parte-3/>. Acesso em: 14 de Nov. de 2014.

OLIVEIRA, Kaui Aires. PostgreSQL x MySQL. Qual Escolher?. Devmedia. 2006. Disponível em: <http://www.devmedia.com.br/postgresql-x-mysql-qual-escolher/3923>. Acesso em: 10 de Nov. de 2014.

ZABOT, J. M.; SILVA, L. C. M. Gestão do Conhecimento: aprendizagem e tecnologia: construindo a inteligência coletiva. São Paulo. Editora Atlas, 2002.