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

FELIPE LIMA

DESENVOLVIMENTO DE UM SISTEMA DE HELP DESK PARA A PREFEITURA MUNICIAL DE CASEIROS - RS

VANESSA LAGO MACHADO
JORGE LUIS BOEIRA BAVARESCO

PASSO FUNDO
2019

FELIPE LIMA

DESENVOLVIMENTO DE UM SISTEMA DE HELP DESK PARA A PREFEITURA MUNICIAL DE CASEIROS - RS

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: Ma. Vanessa Lago Machado

Co-orientador: Me. Jorge Luis Boeira Bavaresco

PASSO FUNDO

2019

FELIPE LIMA

DESENVOLVIMENTO DE UM SISTEMA DE HELP DESK PARA A PREFEITURA MUNICIAL DE CASEIROS - RS

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

Banca Examinadora:

_____

Profª. Ma. Vanessa Lago Machado

Professora Orientadora

_____

Prof. Me. Jorge Luis Boeira Bavaresco

Professor Co-orientador

_____

Profª. Ma. Carmen Vera Scorsatto

_____

Prof. Me. Maikon Cismoski dos Santos

_____

Coordenação do Curso

PASSO FUNDO

2019

DEDICATÓRIA

A minha família,
Pela educação, apoio e incentivo nesta caminhada.
Aos Professores Jorge Luis Boeira Bavaresco e Vanessa Lago Machado,
Pelos conhecimentos e ensinamentos passados,
que contribuíram muito para a realização deste trabalho.
A todos os professores,
pelos ensinamentos passados durante o curso.

EPÍGRAFE

“Só se pode alcançar um grande êxito
quando nos mantemos fiéis a nós mesmos.”
Friedrich Nietzsche

RESUMO

Com o avanço da tecnologia nos dias atuais, as empresas precisam manter-se continuamente atualizadas, a fim de melhorar seus serviços, tornando-os mais ágeis e confiáveis. Ainda, tendo em vista o crescimento das tecnologias surgem os problemas enfrentados pelos seus usuários, aumentando significativamente o serviço de suporte técnico relacionado à área de TI. Considerando tal problema, verifica-se a necessidade de utilização de ferramentas para controlar os atendimentos realizados pela equipe de TI, uma das ferramentas úteis nessa situação são os sistemas *help desk*. Existem diversos sistemas *help desk* disponíveis para utilização, os quais tratam de forma genérica o controle de chamados, mas muitas vezes sistemas genéricos não atendem adequadamente a real necessidade de empresas. Para isso foi proposto e desenvolvido um sistema *help desk* no formato web, com o objetivo de melhorar o controle dos chamados da Prefeitura Municipal de Caseiros, visando melhor atender os usuários e o gerenciamento de equipamentos pertinentes a Prefeitura. O sistema foi desenvolvido na linguagem de programação Java EE com os dados persistidos no banco de dados PostGreSQL por meio da API Hibernate JPA.

Palavras-chave: Help Desk. Desenvolvimento Web. Java EE.

ABSTRACT

With the advancement of technology these days, as companies keep continually focused on their services, becoming more agile and reliable. Still, in view of the growth of technologies arise the problems faced by its users, significantly increasing the service of technical support related to IT area. Considering this problem, there is a need to use tools to control the attendance of the IT staff, one of the useful tools in this situation are the help desk systems. There are several help desk systems available for use, which generally treat call control, but often generic systems do not adequately meet the real business need. To this end, a help desk system in the web format was proposed and developed, aiming to improve the control of calls from the City Hall of Caseiros, aiming to better serve the users and the management of equipment pertinent to City Hall. The system was developed in the Java EE programming language with the data persisted in the PostGreSQL database through the Hibernate JPA API.

Keywords: Help Desk. Web Development. Java EE.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O gerenciamento das atividades dentro de uma empresa é muito importante para o bom andamento dos serviços e atividades nela prestadas. Para tal, os sistemas help desk visam auxiliar no gerenciamento de chamados realizados pelo departamento de Tecnologia da Informação (TI), controlando aberturas de chamados e filas de prioridades de serviços prestados pela TI, como o suporte técnico, além do controle de equipamentos de responsabilidade da TI.

Com base na análise de requisitos realizada na Prefeitura Municipal de Caseiros verificou-se a necessidade de desenvolvimento de um sistema próprio de help desk, visando auxiliar no gerenciamento dos serviços prestados pelo técnico de TI da prefeitura. Essa necessidade surgiu após analisar os sistemas de help desk existentes e disponíveis, em que verificou-se que esses possuem diversas versões genéricas disponíveis para utilização, porém essas versões não refletem adequadamente o cenário individual da prefeitura, visto sua necessidade de controlar manutenções prestadas a equipamentos e os chamados realizados.

O presente trabalho se refere ao desenvolvimento de um sistema de Help Desk, visando melhorar cada vez mais o gerenciamento dos serviços prestados pelo técnico de informática da prefeitura. Assim, o objetivo geral deste trabalho foi desenvolver um sistema Web para resolução de chamados utilizando Java, propondo facilitar ao usuário a solicitação de serviços de suporte na área de TI e melhor controle sobre os chamados e serviços realizados. Tendo como objetivos específicos: gerenciar chamados para o devido controle no atendimento prestado aos usuários; armazenar no banco de dados as informações referentes ao histórico dos chamados; e, gerenciar informações referentes a equipamentos de informática.

Tal sistema foi desenvolvido na linguagem de programação Java EE, sob arquitetura MVC, utilizando a API de persistência do Java, a JPA, para manipulação dos dados presentes no banco de dados PostGreSQL.

Nesse sentido, o trabalho encontra-se organizado em capítulos, em que: o Capítulo 2 aborda o referencial teórico, contendo os principais conceitos acerca de help desk e das tecnologias utilizadas para o desenvolvimento do sistema; o Capítulo 3 aborda os trabalhos correlatos, com uma pesquisa sobre os sistemas de

help desk; O Capítulo 4 aborda a metodologia utilizada, contendo o planejamento para implementação do sistema help desk; o Capítulo 5 aborda o desenvolvimento do sistema, apresentando as fases necessárias para o desenvolvimento, detalhando-as; e, por fim as Considerações Finais e as Referências utilizadas.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos referentes as tecnologias necessárias no desenvolvimento do presente trabalho, para isso constam o estado da arte acerca dessas tecnologias.

2.1 JAVA

Java é uma linguagem de programação orientada a objetos. Inspirada em C e C++, teve seu inicio em 1991, pela Sun Microsystems, com um grupo de pesquisadores da empresa, em um projeto interno, denominado Green Project, com o engenheiro chefe James Gosling assumindo sua coordenação (DEITEL & DEITEL, 2010). De acordo com Luckow e Melo (2010), atualmente a Oracle é a principal mantenedora dessa linguagem de programação.

Java é uma linguagem de programação multiplataforma, assim, para que isso fosse possível os programas em Java são interpretados por uma máquina virtual, própria do Java, a *Java Virtual Machine* (JVM), em que o código é compilado e são gerados os arquivos ".class" das classes que contém os *bytecodes*, os quais são interpretados pela JVM (DEITEL, 2002).

De acordo com a Oracle (2012), para tornar possível a execução dos programas desenvolvidos em Java multiplataforma existem algumas plataformas de desenvolvimento, as quais referem-se ao ambiente particular em que aplicações Java são executadas, são elas: *Standard Edition* (SE), *Enterprise Edition* (EE) e *Micro Edition* (ME). Este trabalho terá como foco a plataforma Java EE.

2.2 JAVA EE

Segundo Gonçalves (2011), o Java EE possui um conjunto de especificações destinadas a aplicações empresariais, essa plataforma foi lançada em 1999 para facilitar o desenvolvimento de aplicações distribuídas, visto que as companhias enfrentavam dificuldades no desenvolvimento de componentes distribuídos, como os

Enterpise Java Beans (EJBs), onde o servidor fica encarregado de gerenciar as instâncias das classes.

Fazem parte também do Java EE serviços como *Transaction* API [1](JTA), *Java Persistence* API (JPA), *Java Naming and Directory Interface* (JNDI), serviços web, entre outros (GONÇALVES, 2011).

2.3 JPA

A JPA proporciona meios de armazenar os dados dos objetos implementados no sistema dentro das tabelas no banco de dados. Para isso, utiliza o modelo objeto-relacional para fazer o mapeamento das tabelas no banco de dados. Oferece uma camada de persistência que permite pesquisar, inserir, remover e atualizar objetos.

A comunicação entre a aplicação e o banco de dados não é realizada diretamente, passando pela API, que faz o mapeamento dessas entidades por meio de anotações, assim o desenvolvedor não precisa preocupar-se com códigos específicos para manipular os dados armazenados (JPA... 2015).

2.4 JSF

O *JavaServer Faces* (JSF) é a especificação para um *framework* de componentes para desenvolvimento web em Java. Essa especificação foi definida por meio da comunidade *Java Community Process* (JCP). Essa comunidade tem como objetivo especificar a evolução da linguagem Java de acordo com o interesse do mercado, e não apenas da Oracle (LUCKOW e MELO, 2010).

Para implementar as funções do JSF é necessário utilizar suas implementações, em que as mais conhecidas são o Mojarra (http://javaserverfaces.java.net), a qual foi usada no desenvolvimento deste trabalho, e o MyFaces da Apache (http://myfaces.apache.org). Utilizando uma implementação é possível criar interfaces de aplicações Java para Web, visto que essas já possuem componentes prontos para o uso. Desse modo o desenvolvedor não se envolve em

[1] *Application Programming Interface* (API), traduzido como “Interface de Programação de Aplicativos".

programar com Javascript e HTML, por exemplo, basta adicionar os componentes, como calendários, tabelas, formulários, e eles serão renderizados e exibidos em formato HTML (GEARY e HOSTMANN, 2012).

2.5 UML

De acordo com Booch, Rumbaugh e Jacobson (2010), a *Unified Modeling Language* (UML) é uma linguagem para modelagem orientada a objetos. Tem como objetivo representar graficamente, por meio de diagramas, a modelagem de um sistema e a comunicação entre os objetos. Surgiu com a necessidade de padronizar os processos de modelagem de *software*.

Os métodos de modelagem que ganharam mais destaque foram os desenvolvidos por Ivar Jacobson (*Object Oriented Software Engineering* - OOSE), Grady Booch (the Booch Method) e James Rumbaugh (*Object Modeling Technique* - OMT) (BOOCH, RUMBAUGH e JACOBSON, 2010).

Ainda, os autores informam que sua primeira versão foi lançada em 1995 com a junção dos métodos Booch e OMT. Mais tarde foi incorporado à linguagem o método OOSE, e em 1996 foi lançada sua versão 0.9. Então com a ajuda de empresas parceiras foi lançada a versão 1.0 com a linguagem bem definida e poderosa, podendo ser aplicada a vários tipos de problemas.

3 SISTEMA HELP DESK

Neste capítulo será apresentado o conceito referente a sistemas de help desk, bem como alguns trabalhos correlatos ao presente trabalho, visando conhecer e comparar alguns sistemas disponíveis.

3.1 O QUE SÃO SISTEMAS DE HELP DESK

Os sistemas de Help Desk (traduzido como “balcão de ajuda“) é um termo da língua inglesa que designa o serviço de apoio a usuários para suporte e resolução de problemas técnicos, os quais podem estar relacionados a tecnologias de informação. Este apoio pode ser tanto dentro de uma empresa (profissionais que cuidam da manutenção de equipamentos e instalações dentro da empresa), quanto externamente (prestação de serviços a usuários), por meio de um sistema de gerenciamento de incidentes ou *call center* (HELPDESK, 2000). O serviço de *help desk* pode ser realizado tanto pessoalmente, quanto por telefone, e-mail, fórum e outros meios de comunicação.

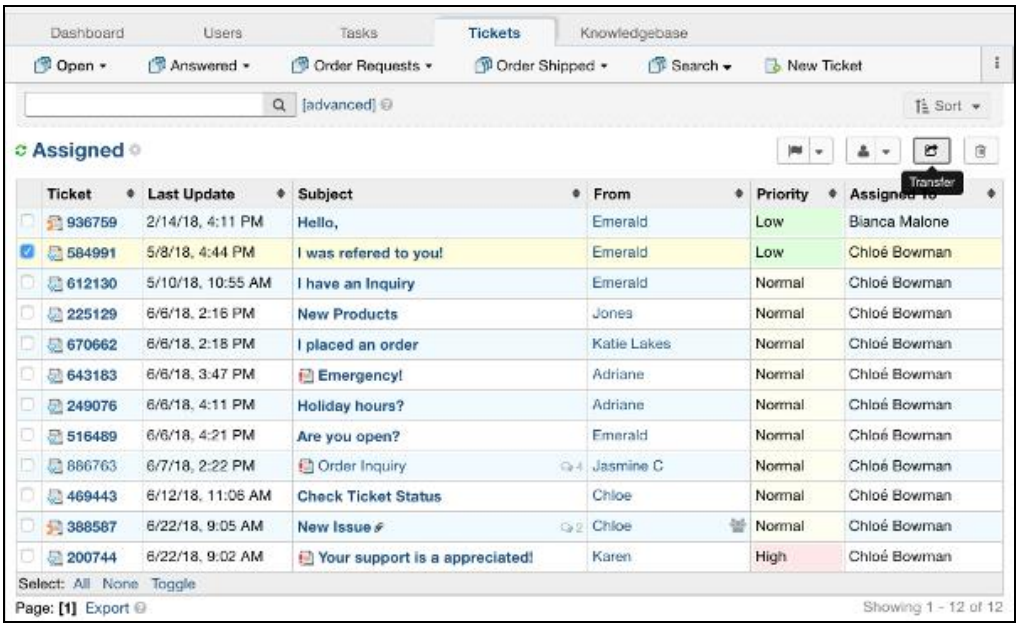
3.2 SISTEMAS CORRELATOS

Nesta seção são descritos alguns trabalhos relacionados ao trabalho desenvolvido, descrevendo as funcionalidades de cada sistema e uma comparação que justifica a necessidade deste trabalho.

3.2.1 OSTICKET - *SUPPORT TICKET SYSTEM*

O sistema de *help desk* osTicket, *Support Ticket System*, refere-se a um sistema de *help desk* customizável, o qual possui uma versão gratuita e uma versão paga. O sistema realiza o gerenciamento de chamados por meio de um sistema web, o qual é utilizado como uma ferramenta de atendimento empresarial (Figura 1) (OSTICKET - SUPPORT TICKET SYSTEM, 2019).

Figura 1 – Tela do sistema de *Help desk* OsTickets, que lista informações do nº do Ticket, data da última atualização, assunto, origem, prioridade e técnico responsável.



Fonte: Osticket - Support Ticket System (2019)

Por padrão o sistema funciona na linguagem inglesa, mas é possível alterar a linguagem, disponível também em português, do Brasil. Em relação ao seu funcionamento, possui três níveis hierárquicos de usuários: o administrador, o agente e o usuário final. O Administrador é responsável por fazer o cadastramento dos agentes e liberação das permissões de cada um, também define as configurações do sistema. O agente é responsável pelo atendimento e resolução dos chamados e podem cadastrar novos usuários. O usuário final possui acesso à solicitação de chamados.

Segundo Hostnet (2018) algumas características do sistema osTicket são:

- **Suporte por E-Mail ou via WEB** – Possibilidade de criação de chamado via e-mail, formulários on-line ou por telefone, em que o chamado é criado pelo próprio profissional de TI;
- **Respostas Automáticas** – Possibilidade de interação automática com os usuários, na abertura de novos chamados ou no recebimento de mensagens;
- **Respostas Prontas** – Possibilidade de elaboração de respostas prontas para questões frequentes;

- **Mensagens Internas** – Possibilidade de envio de mensagens diretas à equipe de suporte;
- **Tópicos de Ajuda** – Possibilidade de configuração de tópicos de ajuda para chamados;
- **Alertas e Notícias** – Possibilidade de envio de alertas por e-mail;
- **Controle de Acesso** – Agentes e usuários são controlados por meio de perfis de acesso baseados em grupos e departamentos;
- **Criação e Transferência de Chamados** – Possibilidade de criação de chamados junto à equipe de suporte e/ou departamento específico;
- **Histórico de Chamados** – Todas as requisições realizadas e respostas são armazenadas no sistema.

3.2.2 TOMTICKET

O sistema *help desk* TomTicket (Figura 2) é uma plataforma online que permite o gerenciamento de chamados e é realizado por meio de um sistema web (tanto para atendentes e clientes), com acesso via dispositivos móveis, disponível aos Sistemas Operacionais (SO) Android e iOS. Há também uma versão desktop, disponível para o perfil de atendente (TOMTICKET, 2018).

Figura 2 – Tela do sistema de *Help desk* TomTicket

Fonte: Tomticket (2018)

O sistema TomTicket permite a geração de relatórios e gráficos diversos para visualização e controle dos atendimentos. Para tal gerenciamento, o sistema possui uma versão gratuita e uma versão paga, em que a versão gratuita é limitada a existência de apenas um atendente e um departamento.

De acordo com Tomticket (2018), alguns recursos desse sistema são:

- Painel do Cliente para Criação e Acompanhamento de Chamados;
- Envio de Anexos pelo Cliente;
- Chamados Internos entre Atendentes;
- Categorização de Clientes via Organizações;
- Cadastro Ilimitado de Clientes;
- Alocar Atendente por Departamento;
- Relatórios de Categoria do Chamado;
- Relatórios de Clientes por Chamado;
- Grupos de Permissão.

3.2.3 ANÁLISE COMPARATIVA DOS SISTEMAS DE HELP DESK

Com base na pesquisa realizada verificou-se que o sistema osTicket não possibilita o gerenciamento dos equipamentos, o que é uma função fundamental aos requisitos elencados para o sistema proposto.

O sistema TomTicket, em sua versão gratuita, permite apenas um atendente e um departamento, sendo que a Prefeitura Municipal de Caseiros, estudo de caso analisado, possui diversos departamentos, e por se tratar de uma entidade pública, a utilização da versão paga é inviável, além disso, esse sistema também não possui um controle de equipamentos de informática.

Desse modo, com base nos requisitos elencados e os sistemas analisados e disponíveis no mercado, verificou-se que nenhum desses atende a todos os requisitos fundamentais para o suporte técnico da Prefeitura.

4 METODOLOGIA

Este capítulo apresenta o planejamento para execução do projeto e implementação do estudo de caso, desde a descrição do contexto atual, levantamento de requisitos do sistema e modelagem dos diagramas.

4.1 ESTUDO DE CASO

O estudo de caso foi realizado sob a demanda vigente na Prefeitura Municipal de Caseiros, que conta com várias repartições pela cidade, incluindo a única escola municipal da cidade que possui dois laboratórios de informática. Nesse cenário, ao todo são mais de 100 computadores e outros equipamentos de informática para prestar suporte técnico. Os computadores são todos conectados a mesma rede interna, portanto em situações de ausência de conexão externa à internet, o funcionamento do sistema proposto não ficará comprometido.

4.2 CENÁRIO ATUAL

Atualmente o suporte técnico aos equipamentos de informática da Prefeitura Municipal de Caseiros não possui controle de chamados, referente aos serviços realizados. Assim, para solicitar a assistência técnica, os usuários precisam ligar para a telefonista para então transferir a ligação para o técnico responsável. Caso não se encontre no momento, o técnico ao receber o recado retorna a ligação. Quando possível à assistência técnica é realizada via telefone ou por acesso remoto, caso não seja possível, o técnico se desloca e realiza a assistência no local da ocorrência. Ainda, salienta-se que o setor de TI conta com apenas um técnico de informática, responsável por para prestar suporte a todas as repartições da prefeitura.

Outra demanda a ser atendida pelo sistema desenvolvido refere-se ao controle de manutenção de equipamentos, com histórico de consertos realizados, e o registro do setor em que se encontra. Atualmente esse controle fica a cargo do

setor de TI, e devido a grande demanda de serviço não há real controle dessas informações.

4.3 DESCRIÇÃO DO NOVO SISTEMA

O sistema desenvolvido neste trabalho tem por finalidade o gerenciamento de chamados e equipamentos de informática da Prefeitura Municipal de Caseiros. Desse modo, o sistema tratará os diferentes perfis dos utilizadores do sistema: administrador, supervisor e usuário. O perfil “administrador” é o técnico que é responsável pela manutenção do sistema e execução dos chamados, o perfil de “supervisor” possui acesso ao sistema para monitoramento de todos os chamados registrados no sistema, e o perfil de “usuário” possui acesso ao sistema para abertura do chamado, quando necessitar de assistência referente a área de TI. Para controlar esses diferentes tipos de acessos o sistema trabalhará com sessões, sendo necessário o cadastro prévio e o login no sistema.

4.4 REQUISITOS FUNCIONAIS

- Realizar login.
- Gerenciar usuários.
- Usuário solicitar chamado.
- Gerenciar equipamentos de informática.
- Gerenciar peças/equipamentos em conserto.
- Enviar e-mail.

4.5 REQUISITOS NÃO-FUNCIONAIS

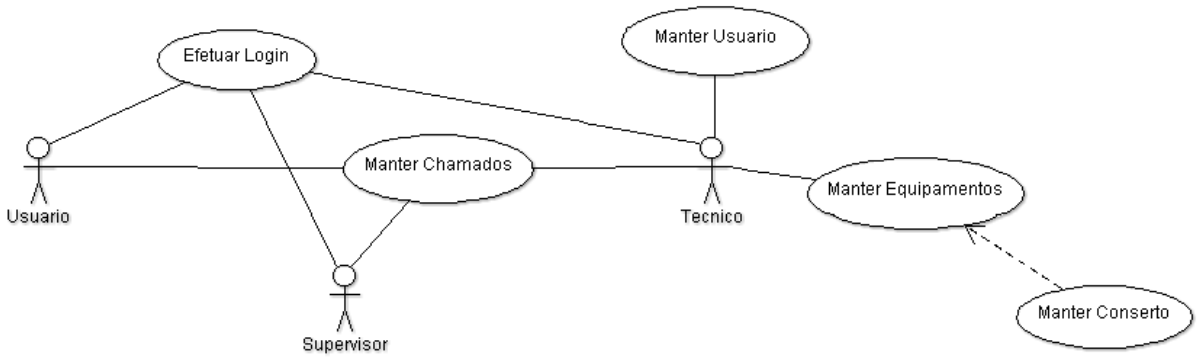
- Linguagem Java.
- Banco de dados PostgreSQL.

4.6 DIAGRAMA DE CASO DE USO

No diagrama de caso de uso, ilustrado na Figura 3, verifica-se as funcionalidades de cada ator do sistema, realizadas após o login, em que:

- O usuário ao logar no sistema poderá realizar uma solicitação de chamado e também fazer uma consulta de suas solicitações.
- O supervisor poderá realizar abertura de chamado e também monitorar o andamento de todos os chamados, via sistema ou a partir da emissão de relatórios.
- Para o técnico será exibida a tela com os chamados pendentes, para gerenciar os chamados, o técnico realiza os atendimentos, e após solucioná-los finaliza os mesmos no sistema. Ficará encarregado de cadastrar os usuários e definir suas permissões, gerenciar os equipamentos, incluindo o controle dos que encontram-se em manutenção, além de gerar relatórios.

Figura 3 – Diagrama de casos de uso do sistema de *help desk* da Prefeitura Municipal de Caseiros

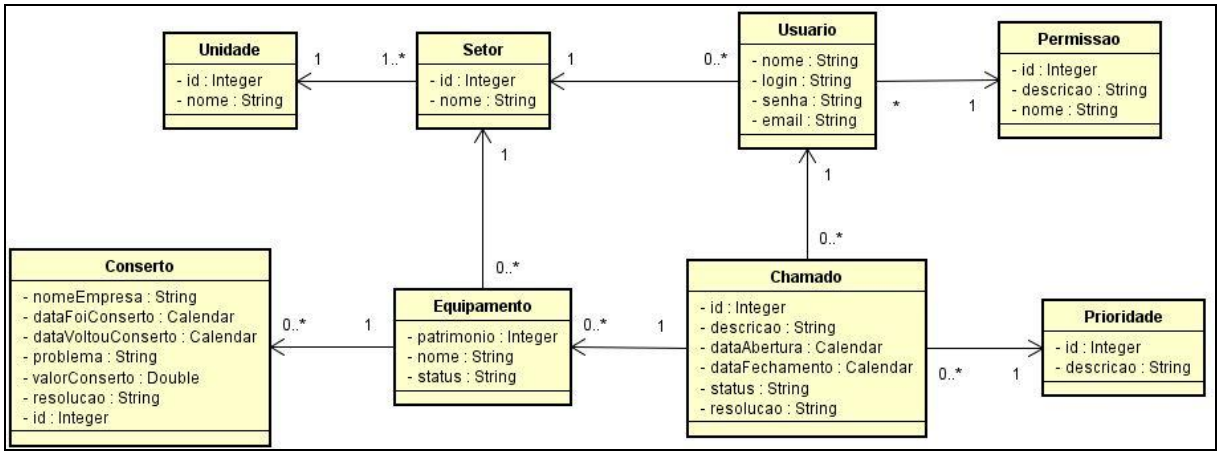


Fonte: Do Autor

4.7 DIAGRAMA DE CLASSES

Na Figura 4 encontra-se ilustrado o diagrama de classes do sistema implementado, representando as relações entre os objetos.

Figura 4 – Diagrama de classes do sistema de *help desk* da Prefeitura Municipal de Caseiros



Fonte: Do Autor

Desse modo, a classe “Usuario” conta com os atributos login e senha para autenticação no sistema, além dessas informações é armazenado o nome, e-mail, setor, e suas permissões, sendo o nome para identificar a pessoa responsável pela solicitação de assistência, o e-mail é necessário para acompanhamento da resolução dos chamados, e o setor é utilizado para identificar em que setor o usuário encontra-se lotado. Dessa forma, a classe “setor” representa os setores existentes na prefeitura. A classe “Permissoes” realiza a atribuição de acessos, garantindo as permissões de acesso que cada usuário poderá realizar no sistema. A classe “Chamado” possui os atributos “dataAbertura” e “dataFechamento”, um atributo “status”, que serve para verificar o andamento do chamado, um atributo “usuario”, para manter a informação referente a qual o usuário que realizou a abertura do chamado, e um atributo “prioridade”, que será utilizado para identificar quais chamados possuem uma prioridade mais alta, e necessitam ser resolvidos antes.

A classe “Equipamento” modela os dados referentes aos equipamentos de informática, tais como: computadores, impressoras e peças de reposição, os quais possuem um número de patrimônio, um nome (descrição do equipamento) e seu status, e tais equipamentos encontram-se situados em determinado setor.

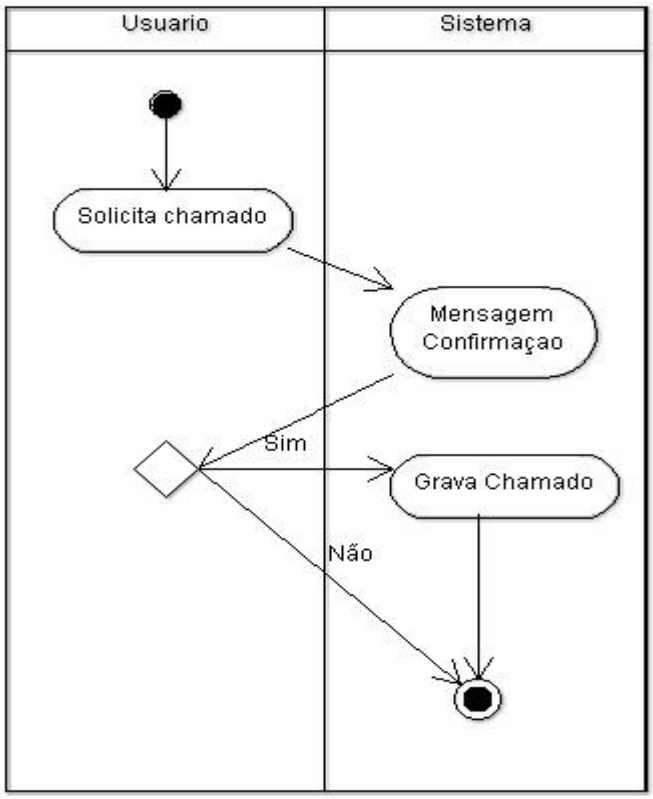
A classe “Conserto” modela os dados de quais equipamentos foram encaminhados a conserto terceirizado, para isso são armazenados dados do nome da empresa responsável pelo conserto, data em que o equipamento foi encaminhado ao conserto e a data que o mesmo retornou, problema identificado, valor do conserto e resolução.

Por fim, a classe “Prioridades” determina as prioridades possíveis e sua ordem, utilizada pelo técnico para decidir a ordem de necessidade de atendimento aos chamados, atendendo as urgências pré-definidas.

4.8 DIAGRAMA DE ATIVIDADES

O usuário responsável pela abertura de chamados, quando realizado pelo usuário final, ao logar no sistema terá as opções de solicitar a abertura de um chamado ou consultar seus chamados já realizados, ao solicitar a abertura de um chamado, descrevendo o problema e sua urgência, o sistema solicitará uma mensagem para confirmação, para que o usuário possa confirmar sua solicitação, e com isso o chamado é registrado no banco de dados, conforme ilustrado no diagrama de atividades do perfil de usuário (Figura 5).

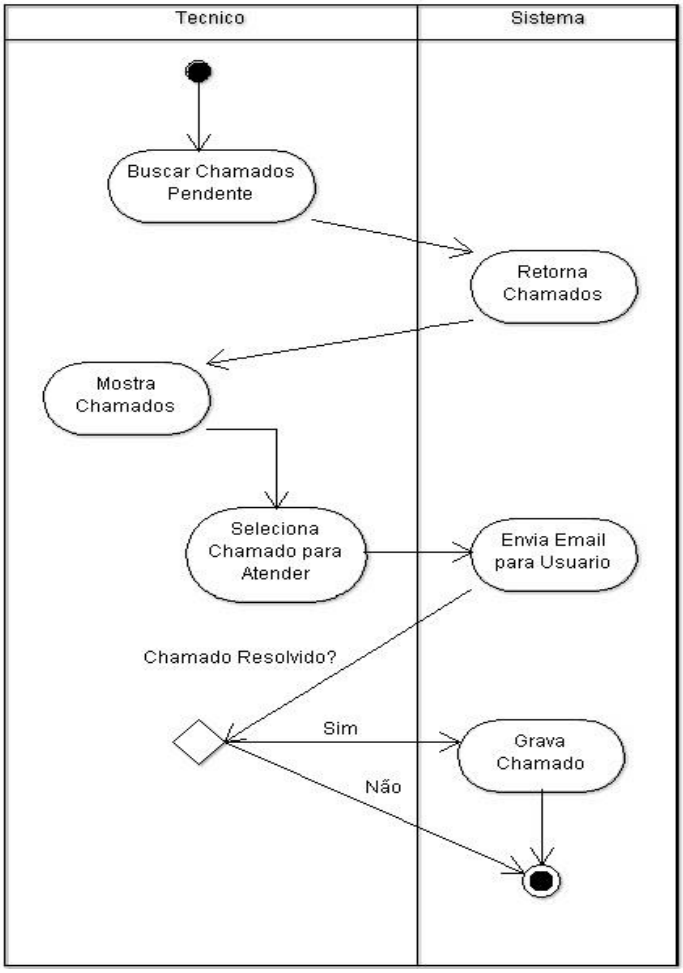
Figura 5 – Diagrama de atividades do perfil de “usuário” do sistema de *help desk* da Prefeitura Municipal de Caseiros



Fonte: Do Autor

Quando o técnico logar no sistema, os chamados pendentes são listados na tela principal. Assim, se houver chamados pendentes o técnico selecionará um chamado para atendê-lo, com isso um e-mail é enviado ao usuário informando-o sobre a abertura do chamado. Assim, após atender ao chamado, se solucionado o técnico irá informar o fechamento do chamado, caso contrário o chamado retornará para a lista de chamados pendentes. Tal fluxo encontra-se ilustrado na Figura 6, referente ao diagrama de atividades do perfil de técnico.

Figura 6 – Diagrama de atividades do perfil de "técnico" do sistema de *help desk* da Prefeitura Municipal de Caseiros



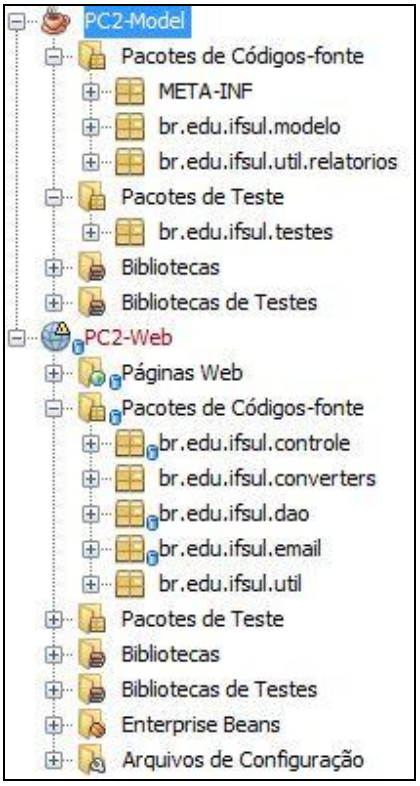
Fonte: Do Autor

5 DESENVOLVIMENTO

Para o desenvolvimento do novo sistema foi utilizando o padrão de arquitetura de desenvolvimento Modelo, Visão e Controle (MVC), que consiste em dividir o projeto em três camadas, onde a camada de modelo é responsável pela leitura e escrita dos dados e suas validações. A camada de visão é responsável pela interação com o usuário, onde os dados são exibidos ao usuário e o mesmo pode inserir informações no sistema. A camada de controle serve como intermediária entre as outras camadas, recebendo as requisições e dados da camada visão e realizando as manipulações dos dados para acesso ao banco de dados.

Para isso o desenvolvimento foi dividido na implementação de dois projetos. No primeiro projeto (nomeado de **PC2-Model)** foi desenvolvido a camada de modelo e no segundo (nomeado de **PC2-Web)** foram desenvolvidos as camadas de controle e visão. Os projetos foram desenvolvidos utilizando a IDE NetBeans, na versão 8.2. A Figura 7 mostra a estrutura dos projetos.

Figura 7 – Estrutura dos projetos realizados na IDE NetBeans para implementação do sistema de *help desk* da Prefeitura Municipal de Caseiros



Fonte: Do Autor

Para a persistência dos dados em um banco de dados foi utilizado o Sistema Gerenciador de Banco de Dados (SGBD) PostgreSql, utilizando a ferramenta pgAdminIII.

5.1 CAMADA MODELO

A camada de Modelo é responsável pela modelagem do sistema, a qual foi implementada seguindo o diagrama proposto na Figura 4, além disso, nesse caso a modelagem é replicada para o banco de dados por meio de persistência de dados. Desse modo, foram utilizadas algumas bibliotecas e arquivos de configuração para a persistência com o banco de dados, em que as bibliotecas utilizadas foram: a Hibernate JPA, na versão 4.3.11, utilizada para realizar a persistência com o banco de dados no modelo objeto-relacional; a biblioteca Hibernate Validator, na versão 5.2.4, utilizada para validação das informações dos objetos persistidos; além do driver do PostgreSQL para conexão da aplicação com o banco de dados, possibilitando manipulação e manutenção das tabelas.

Além disso, faz-se necessário algumas configurações referentes a persistência de dados, em que são informadas as informações pertinentes à comunicação com o banco de dados, informando qual é a base de dados que será utilizada na comunicação, bem como os dados de acesso. Nesse arquivo é possível informar algumas configurações extras, como a definição de que a atualização das tabelas do banco de dados será realizada de forma automática.

As classes modeladas seguem o padrão de desenvolvimento JavaBeans (também chamados apenas de *beans*) que são componentes utilizados para criação de aplicações e fornecem grande flexibilidade pois permitem ao programador reutilizar e integrar diferentes componentes existentes. Para isso as classes implementam a interface Serializable, usada para serializar os dados transformando em um vetor de *bytes* para serem transportados (pela rede, disco, etc) e depois desserializá-los. Além disso, todos os atributos devem estar devidamente encapsulados. Tal modelagem encontra-se ilustrada na Figura 8.

Figura 8 – Padrão de modelagem de classe do sistema de *help desk* desenvolvido, exemplo da Classe Chamado.

```
@Entity
@Table(name = "chamado")
public class Chamado implements Serializable{

    @Id
    @SequenceGenerator(name = "seq_chamado", sequenceName = "seq_chamado_id", allocationSize = 1)
    @GeneratedValue(generator = "seq_chamado", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @NotNull(message = "A descrição não pode ser nula")
    @NotBlank(message = "A descrição não pode ser em branco")
    @Length(max = 100, message = "A descrição não pode ter mais que {max} caracteres")
    @Column(name = "descricao", length = 100, nullable = false)
    private String descricao;
    @NotNull(message = "O status não pode ser nula")
    @NotBlank(message = "O status não pode ser em branco")
    @Length(max = 100, message = "O status não pode ter mais que {max} caracteres")
    @Column(name = "status", length = 100, nullable = false)
    private String status;
    @Length(max = 100, message = "A resolução não pode ter mais que {max} caracteres")
    @Column(name = "resolucao", length = 100)
    private String resolucao;
    @Temporal(TemporalType.DATE)
    @NotNull(message = "Data de abertura não pode ser nula")
    @Column(name = "dataAbertura", nullable = false)
    private Calendar dataAbertura;
    @Temporal(TemporalType.DATE)
    @Column(name = "dataFechamento")
    private Calendar dataFechamento;
    @Column(name = "finalizado")
    private Boolean finalizado;
    @ManyToOne
    @JoinColumn(name = "prioridade", referencedColumnName = "id", nullable = false,
            foreignKey = @ForeignKey(name = "fk_chamado_prioridade"))
    private Prioridade prioridade;
```

Fonte: Do Autor

Para realizar o mapeamento e persistência das classes no banco de dados são utilizadas anotações que são precedidas pelo símbolo “@”. A anotação @Entity define a classe como uma *entity bean* e será representada no banco de dados pela anotação @*Table*, que conterá o nome da tabela.

A anotação @Id define o atributo como chave-primária da tabela, e é uma anotação obrigatória. A anotação *@SequenceGenerator* será responsável por atribuir em uma sequência especificada um valor para a chave-primária, definido pelo atributo *allocationSize.* A especificação da sequência dar-se-a pela anotação *@GeneratedValue,* em que neste trabalho foi utilizado a estratégia *SEQUENCE*, gerando um valor sequencial para a chave primária, não sendo necessário informar manualmente esse valor.

Ainda, relacionado à persistência dos dados, utiliza-se a anotação @Column vinculado ao atributo da classe para definir e configurar a coluna no banco dados, com o qual é possível definir o nome, tipo e restrições da coluna na tabela.

Para validação dos campos, baseado nas restrições do banco de dados, pode-se utilizar algumas anotações, como: *@NotNull*, informando uma mensagem de validação no caso de receber um valor nulo; *@NotBlank*, para validar campos em relação a atributos com valores em branco; *@Length*, que validará o limite de caracteres que o atributo pode receber. Para tratamento dos campos que armazenam data e hora utiliza-se a anotação *@Temporal*.

Para tratar o relacionamento entre as classes, modelando-os para a persistência de dados, utilizam-se anotações, em que a anotação *@ManyToOne* é utilizada para modelar uma relação do tipo muitos para um, relacionando duas tabelas vinculadas com chave estrangeira, para isso utiliza-se a anotação @JoinColumn que é responsável por fazer a referência com a chave primária da tabela referenciada, conforme exemplo disposto nas Figuras 9 e 10, em que a Figura 9 apresenta o segmento de código da classe Chamado, da camada de modelo, referente ao relacionamento muitos para um, com a classe Usuario.

Figura 9 – Relacionamento ManyToOne na classe Chamado, da camada de modelo do desenvolvimento do sistema de *help desk*.

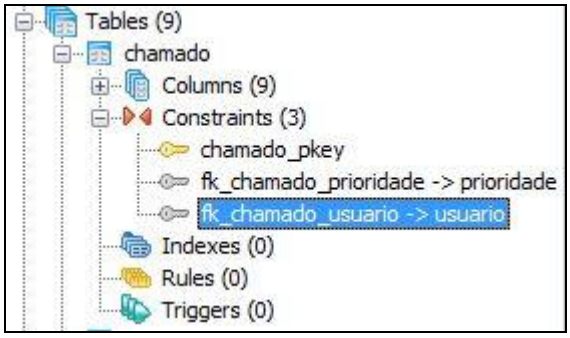
```
@ManyToOne
@JoinColumn(name = "usuario", referencedColumnName = "login", nullable = false,
        foreignKey = @ForeignKey(name = "fk_chamado_usuario"))
private Usuario usuario;

public Chamado() {
}
```

Fonte: Do Autor

Com base na persistência de dados, realizadas com a API JPA, a tabela chamado é gerada no banco de dados contendo a chave estrangeira para a relação com a tabela Usuario, conforme apresentado na Figura 10.

Figura 10 – Chave Estrangeira no Banco de Dados da classe Chamado do relacionamento ManyToOne, da camada de modelo do desenvolvimento do sistema de *help desk*.



Fonte: Do Autor

O relacionamento um para muitos é representado nos casos em que é necessário ter acesso, por meio da modelagem, a todas as informações envolvendo a relação, inclusive de todos os itens que compõe essa relação. Como pode ser visto neste trabalho, na relação entre equipamento e conserto, em que um equipamento pode ter sido consertado inúmeras vezes e de cada equipamento faz-se necessário ter a informação de todos os consertos realizados, sendo necessário que a relação seja evidenciada na modelagem. Tal exemplo segue evidenciado nas Figuras 11, 12 e 13, em que a Figura 11 apresenta o segmento de código da classe Equipamento, apresentando o mapeamento do relacionamento entre equipamento e conserto.

Figura 11 – Segmento de Código da classe Equipamento sobre o Relacionamento OneToMany.

```
@OneToMany(mappedBy = "equipamento", cascade = CascadeType.ALL,
        orphanRemoval = true, fetch = FetchType.LAZY)
private List<Conserto> consertos = new ArrayList<>();

public Equipamento() {
}
```

Fonte: Do Autor

Do mesmo modo, a Figura 12 apresenta o segmento de código da classe Equipamento, referente ao mapeamento do relacionamento de conserto com equipamento.

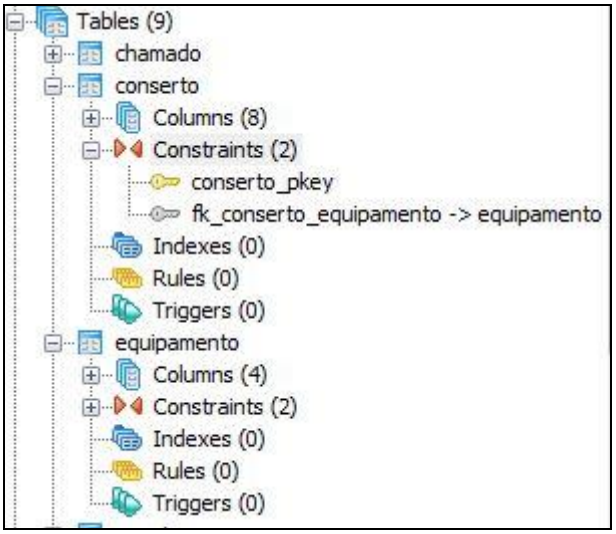
Figura 12 – Segmento de Código da classe Conserto sobre o Relacionamento ManyToOne.

```
@NotNull(message = "O Equipamento não pode ser nulo")
@ManyToOne
@JoinColumn(name = "equipamento", referencedColumnName = "patrimonio", nullable = false)
@ForeignKey(name = "fk_conserto_equipamento")
private Equipamento equipamento;
```

Fonte: Do Autor

Por fim, com base nos mapeamentos realizados na camada de modelo, com a utilização da API de persistência de dados JPA, as tabelas são geradas no banco de dados, contendo o devido relacionamento dos mapeamentos gerados nas classes correspondentes. Desse modo, a Figura 13 apresenta as tabelas conserto e equipamento no banco de dados, em que a tabela conserto contém a chave estrangeira referenciando o equipamento previamente cadastrado.

Figura 13 – Tabelas Conserto e Equipamento com o devido mapeamento do relacionamento entre as tabelas.



Fonte: Do Autor

O relacionamento de muitos para muitos pode ser observado na relação entre usuário e permissão, onde muitos usuários podem ter varias permissões e as permissões podem ser de muitos usuários, tal relação gera uma nova tabela intermediária. Desse modo a anotação *@JoinColumns* é responsável pelo mapeamento de colunas do lado proprietário. O atributo *name* contém o nome da coluna na tabela intermediária e o atributo *referencedColumnName* contém o nome da coluna da chave primária do lado proprietário. A anotação @inverseJoinColumns faz o mapeamento do lado inverso, conforme visto na Figura 14.

Figura 14 – Segmento de Código da classe Usuario sobre o Relacionamento ManyToMany.

```
@ManyToMany
@JoinTable(name = "permissoes",
        joinColumns =
        @JoinColumn(name = "usuario", referencedColumnName = "login", nullable = false),
        inverseJoinColumns =
        @JoinColumn(name = "permissao", referencedColumnName = "nome", nullable = false))
private Set<Permissao> permissoes = new HashSet<>();

public Usuario(){

}
```

Fonte: Do Autor

Para a criação da tabela intermediária, via persistência de dados, é usada a anotação *@JoinTable*, identificada pelo atributo *name,* que conterá duas chaves estrangeiras, referenciando as chaves primárias de cada uma das tabelas, usuário e permissão, para o mapeamento entre elas. Tal modelagem no banco de dados é apresentada na Figura 15, em que a tabela permissoes foi criada como a tabela intermediaria entre as tabelas “Usuario” e “Permissao”, contendo os atributos de chave primária composta por “usuario” e “permissao”.

Figura 15 – Chaves Estrangeiras no Banco de Dados relacionamento ManyToMany.

Fonte: Do Autor

5.2 CAMADA CONTROLE

A camada de controle é a responsável por realizar a comunicação entre a camada de visão e a camada de modelo. Para isso, é utilizado uma lógica de acesso aos dados (Data Acess Object – DAO), que é responsável por gerenciar essa comunicação, ainda possibilita a reutilização de código, contendo recursos de busca, filtros, ordenação e paginação. Como os métodos se repetem para todos os acessos aos dados, alterando somente o objeto a ser manipulado, foi criado uma classe genérica contendo os métodos (classe DAOGenerico), desse modo, para utilizar o DAO específico à cada classe, basta estender a classe DAOGenerico, e informar, por meio de manipulação de uma variável o nome da classe a ser persistida, essas devem também implementar a interface Serializable.

Além disso, a camada de controle contém os métodos para manipular os dados requisitados pelos componentes da interface gráfica, como listagens e formulários, para isso foi criado uma classe específica para cada controle de objeto.

Nas classes de controle a anotação @Named, identificada pelo atributo *value,* possibilita referenciar esse *bean* dentro uma página JSF. Do mesmo modo é possível informar o tipo de escopo, que os dados serão manipulados, que são: *RequestScoped, SessionScoped, ApplicationScoped e ViewScoped, os quais encontram-se descritos a seguir.*

- ***RequestScoped***: é o escopo padrão caso não seja informado nenhum. Ele funciona como um simples HTTP request, o qual é descartado ao fim de cada requisição, pois o ManagedBean não manterá seu estado entre as solicitações/requisições HTTP que o usuário fizer;
- ***SessionScoped***: Nesse tipo de escopo todos os objetos e atributos vinculados ao ManagedBean sobreviverão durante toda a sessão do usuário, e a sessão é definida pelo vinculo do navegador do usuário com o servidor;
- ***ApplicationScoped***: Refere-se ao escopo que mantém apenas uma única instância na memória em toda a aplicação, em que todos os usuários têm acesso ao mesmo ManagedBean, logo não é aconselhável guardar informações de usuários com esse escopo;

- ***ViewScoped***: Nesse tipo de escopo o estado do bean é mantido enquanto houver requisições da mesma página, e quando ele muda de página ou atualiza a mesma página o estado do bean é descartado.

Um exemplo do uso de escopo na implementação pode ser vista na Figura 16, que apresenta a criação da classe de ControleChamado, com a devida identificação do bean, permitindo referenciar tais dados na página JSF, além da definição de escopo, nesse caso *ViewScoped*.

Figura 16 – Segmento do código da classe ControleChamado, da camada Controle, do desenvolvimento do sistema de *help desk*.

```
/**
 *
 * @author Felipe
 */
@Named(value = "controleChamado")
@ViewScoped
public class ControleChamado implements Serializable {

    @EJB
    private ChamadoDAO<Chamado> dao;
    private Chamado objeto;
    private Boolean editando;

    @EJB
    private UsuarioDAO<Usuario> daoUsuario;
    private Usuario usuario;
    private Boolean novoObjeto;

    private Calendar data = Calendar.getInstance();

    public ControleChamado() {
        editando = false;
    }
```

Fonte: Do Autor

A classe responsável pelo controle implementa os métodos novo, alterar, salvar e finalizar, os quais realizam a interface entre o banco de dados e a interface com o usuário, realizando a ação solicitada pelo usuário, para isso o acesso aos dados no banco de dados é realizada por meio do *DAO* acessíveis nos métodos *persist* e *merge* implementados na classe *DAO*. Ainda, o método *excluir()* busca no

banco de dados o objeto por meio do *id* e executa o método *remove()*, da classe *DAO*, que deleta o registro do banco de dados. A Figura 17 apresenta estes métodos.

Figura 17 – Segmento de código da classe ControleChamado, da camada de Controle, contendo os métodos novo, alterar, salvar e finalizar.

```
public void novo() throws Exception {
    objeto = new Chamado();
    editando = true;
    String status = "Aguardando Técnico";
    objeto.setStatus(status);
    objeto.setFinalizado(false);
}

public void alterar(Object id) {
    try {
        objeto = dao.getObjectById(id);
        editando = true;
    } catch (Exception e) {
        Util.mensagemErro("Erro ao recuperar objeto: " + Util.getMensagemErro(e));
    }
}

public void excluir(Object id) {
    try {
        objeto = dao.getObjectById(id);
        dao.remove(objeto);
        Util.mensagemInformacao("Objeto removido com sucesso");
    } catch (Exception e) {
        Util.mensagemErro("Erro ao remover objeto: " + Util.getMensagemErro(e));
    }
}

public void finalizar(Object id) {
    try {
        objeto = dao.getObjectById(id);
        objeto.setFinalizado(true);
        dao.merge(objeto);
        EnviarEmail email = new EnviarEmail();
        email.enviar(objeto);
```

Fonte: Do Autor

5.2.1 CONVERSORES

As classes de conversores implementam a conversão de dados por meio da interface Converter e dos métodos abstratos *getAsObject* e *getAsString***,** em que o

primeiro recebe por parâmetro uma *String* e retorna um objeto da classe solicitado para o conversor, e o segundo faz a operação inversa, recebe por parâmetro um objeto e retorna uma *String* desse objeto. A Figura 18 apresenta o conversor da classe Chamado.

Figura 18 – Classe ConverterChamado, da camada Controle, do desenvolvimento do sistema de *help desk*.

```
/**
 *
 * @author Felipe
 */
@FacesConverter(value = "converterChamado")
public class converterChamado implements Serializable, Converter{

    @PersistenceContext(unitName = "PC2-webPU")
    private EntityManager em;

    @Override
    public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
        if(string == null || string.equals("Selecione um registro")){
            return null;
        }
        return em.find(Chamado.class, Integer.parseInt(string));
    }

    @Override
    public String getAsString(FacesContext fc, UIComponent uic, Object o) {
        if(o == null){
            return null;
        }
        Chamado obj = (Chamado) o;
        return obj.getId().toString();
    }

}
```

Fonte: Do Autor

5.3 CAMADA VISÃO

A camada de visão foi desenvolvida a partir do *framework* JSF e é responsável por fazer as requisições ao banco de dados e mostrá-los na tela para os usuários, também é onde se encontram os menus, campos para inserir as

informações e os botões para funcionamento do sistema. Seus arquivos das páginas web possuem extensão *.xhtml.*

O arquivo template.xhtml possui o layout do sistema e também as áreas onde serão será apresentadas as informações referentes as requisições de cada página do sistema.

Os menus do sistema são controlados para que somente os usuários com as permissões garantidas possam acessá-los. Esse controle é feito pelo atributo *rendered*, que faz o teste para verificar se o usuário está logado ao sistema e renderiza na tela os menus e os registros conforme a permissão concedida ao usuário.

Para fazer a listagem dos registros do banco de dados foi criado o arquivo *listar.xhtml*, que por meio dos controles da classe informa ao DAO específico qual busca será realizada no banco de dados, conforme a opção desejada.

A inclusão e alteração dos registros no banco de dados é feito pelo arquivo *formulário.xhtml,* que contém os campos para inserção das informações pelo usuário. Que assim como a listagem dos dados, os controladores da classe também informam ao DAO específico qual classe será manipulada.

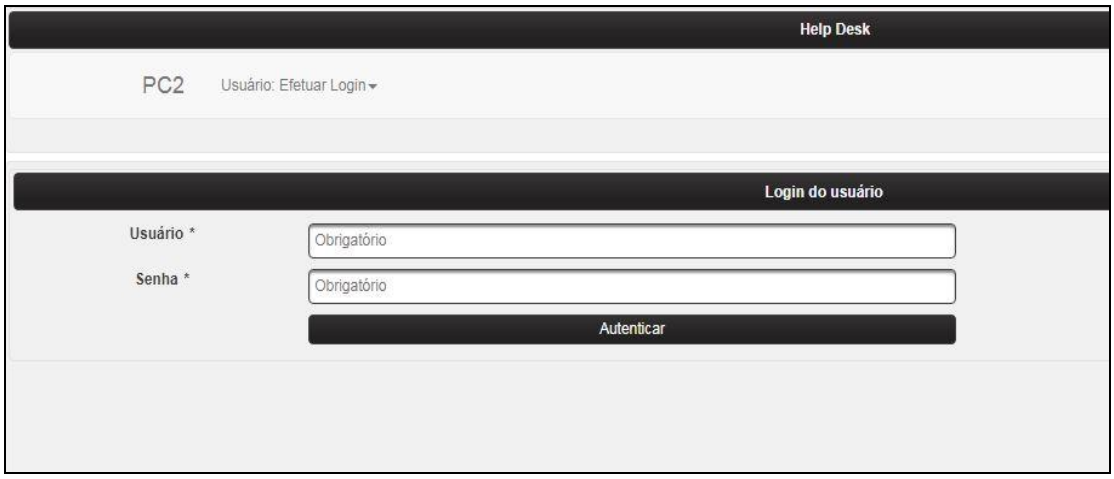
5.4 SISTEMA DE HELP DESK

Nesta seção será apresentada a execução e funcionamento do novo sistema, apresentando as telas geradas pelo sistema, como login, listagem dos dados, inclusão, alteração e exclusão de registros.

5.4.1 LOGIN NO SISTEMA

Ao acessar o sistema será apresentada a tela de login do usuário, Figura 19, permitindo o acesso ao sistema, que após login será direcionado para a página inicial, conforme perfil do usuário.

Figura 19 – Tela de login do sistema de help desk da Prefeitura Municipal de Caseiros.



Fonte: Do Autor

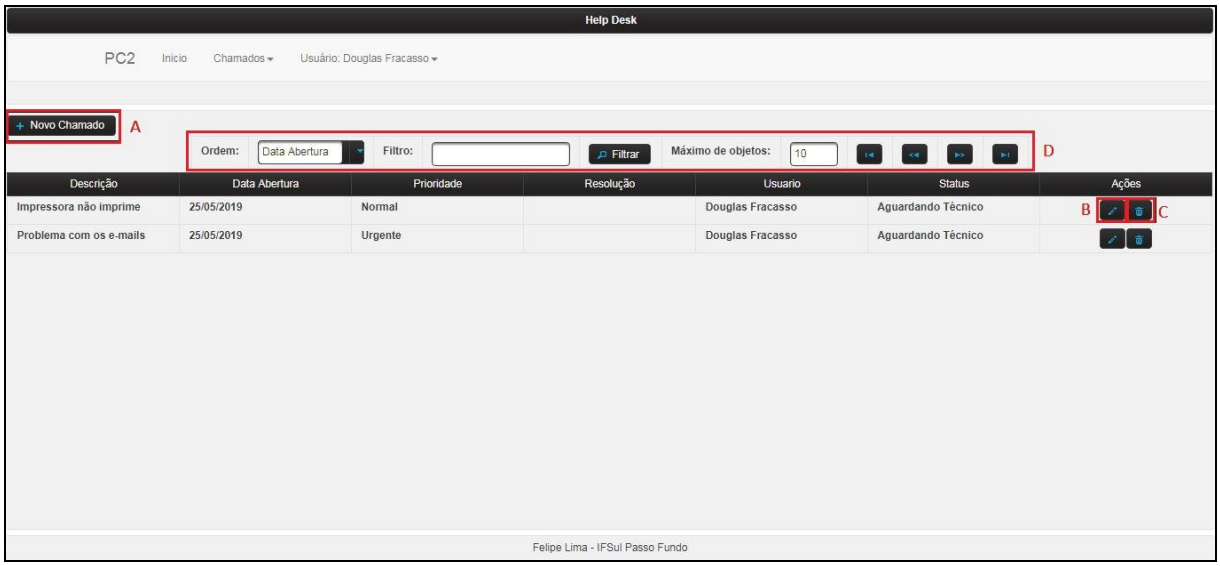
Em relação aos dados de acesso, para login no sistema, essas informações referem-se ao nome do usuário e a senha. Essas informações são previamente concedidas pelo administrador do sistema.

5.4.2 PERFIL DO TIPO “USUARIO”

Ao logar no sistema o usuário com o perfil do tipo “Usuario” será redirecionado para a página de listagem dos chamados realizados e em aberto, conforme apresentado na Figura 20. No primeiro acesso ao sistema, ou quando todos os chamados realizados forem atendidos, a tela inicial (de chamados abertos) retornará uma mensagem informando que não há nenhum registro.

Esse perfil de usuário pode somente realizar um novo chamado, pelo botão “Novo Chamado” (Figura 20-A), que quando realizado, receberá um e-mail avisando-o sobre a abertura do chamado. Além disso, é possível editar um chamado já realizado, representado pelo botão na Figura 20-B, e excluir um chamado realizado, representado pelo botão na Figura 20-C. Ainda, para visualização dos seus chamados, há diversos modos de ordenação, filtros e paginação dos registros disponíveis no sistema, conforme representado na Figura 20-D.

Figura 20 - Tela inicial perfil de "usuario"



Fonte: Do Autor

Em complemento aos seus chamados, esse perfil pode também visualizar os seus chamados finalizados, acessado pelo menu “Chamados” e opção “Finalizados”, como mostra a Figura 21. Essa listagem apresenta os chamados com a data em que foram finalizados e a resolução do problema.

Figura 21 – Fragmento de tela apresentando os menus de acesso dos usuários com perfil “Usuario”, opção chamados finalizados.

Fonte: Do Autor

Além disso, o usuário pode ver seu perfil por meio do menu de “Usuário”, onde é possível alterar algumas informações como e-mail e senha. Ainda nesse menu, possui a opção “Efetuar Logout”, para sair do sistema, representado pela Figura 22.

Figura 22 – Fragmento de tela apresentando os menus de acesso dos usuários com perfil "Usuario", opção Efetuar Logout.

Fonte: Do Autor

5.4.3 PERFIL DO TIPO "SUPERVISOR"

O usuário com perfil de "Supervisor" é o responsável por supervisionar o trabalho do usuário com perfil de "Administrador". Para isso ele possui as mesmas listagens apresentadas para o usuário com perfil de "Usuario", a diferença é que este pode visualizar os chamados em aberto e finalizados de todos os usuários, fazendo o acompanhamento dos trabalhos realizados, isso é possível por meio do menu "Chamados", submenu "Todos" e opções "Em Aberto" ou "Finalizados", conforme representado na Figura 23.

Figura 23 – Fragmento de tela apresentando os menus de acesso dos usuários com perfil "Supervisor".

Fonte: Do Autor

5.4.4 PERFIL DO TIPO “ADMINISTRADOR”

O perfil de “Administrador” refere-se ao técnico do setor de TI, o qual é responsável por realizar os atendimentos dos chamados solicitados e a finalização dos mesmos, assim que o problema for resolvido.

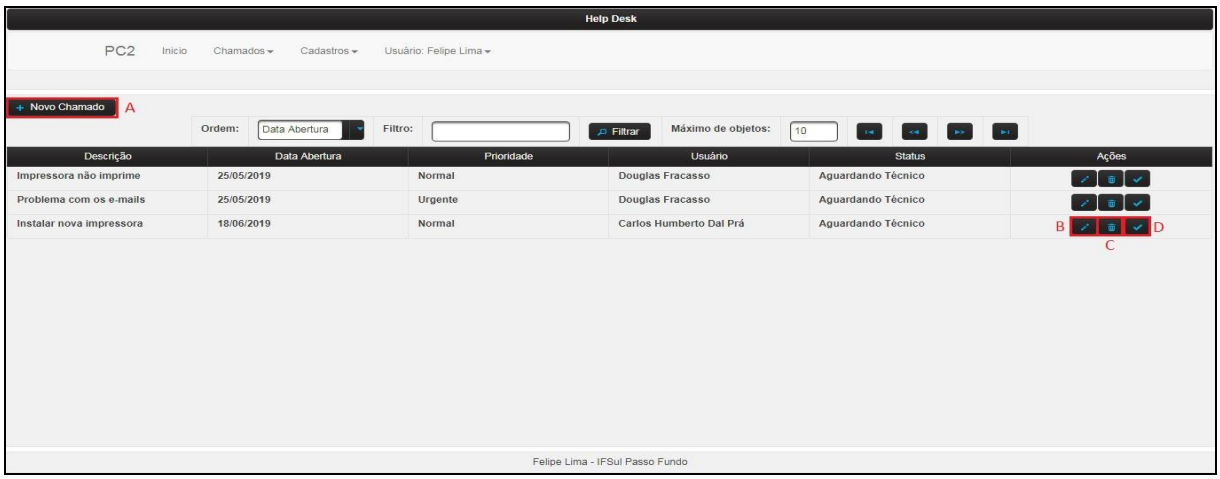
5.4.4.1 TELA INICIAL

Ao logar no sistema será apresentada a tela contendo a listagem dos chamados em aberto, solicitados por todos os usuários. Nessa tela possui um botão de “Novo Chamado” (Figura 24-A), para a necessidade de abrir um chamado para um usuário.

Nessa tela o técnico pode realizar a atualização ou finalização dos chamados, por meio do botão ilustrado na Figura 24-B. Para o caso da finalização do chamado é informado a data de finalização e a resolução do problema, após deve ser confirmado no botão ilustrado na Figura 24-D. Toda movimentação ocorrida no chamado é informada ao usuário por e-mail.

Ainda, é possível realizar a exclusão de um chamado, por meio do botão ilustrado na Figura 24-C.

Figura 24 - Tela inicial perfil "Administrador"

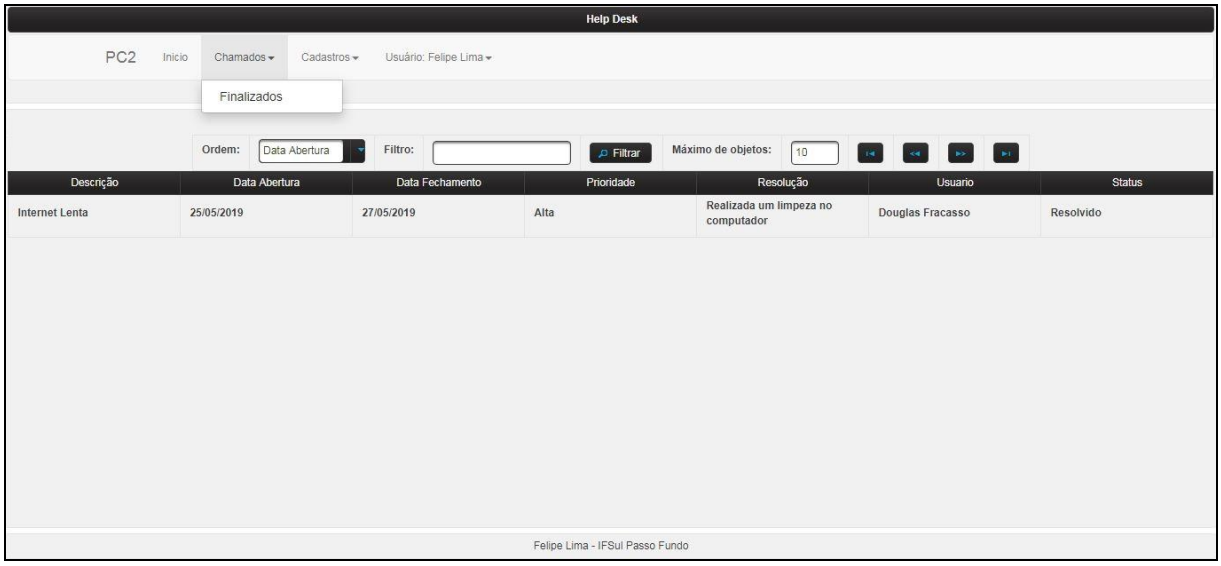


Fonte: Do Autor

5.4.4.2 MENU “CHAMADOS”

Este menu possui a opção “Finalizados” representado na Figura 25, em que são listados os chamados já finalizados de todos os usuário, contendo também as opções de ordenações e filtros.

Figura 25 - Tela chamados finalizados perfil "Administrador"

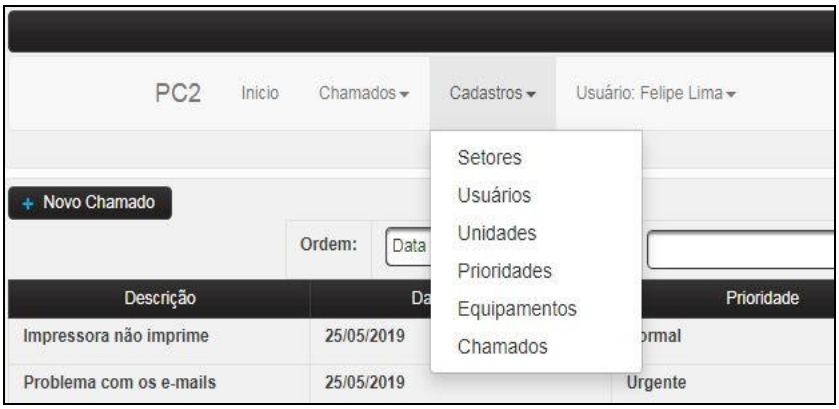


Fonte: Do Autor

5.4.4.3 MENU “CADASTROS”

Neste menu são realizadas as manutenções de setores, unidades, prioridades, equipamentos e usuários do sistema, representado pela Figura 26.

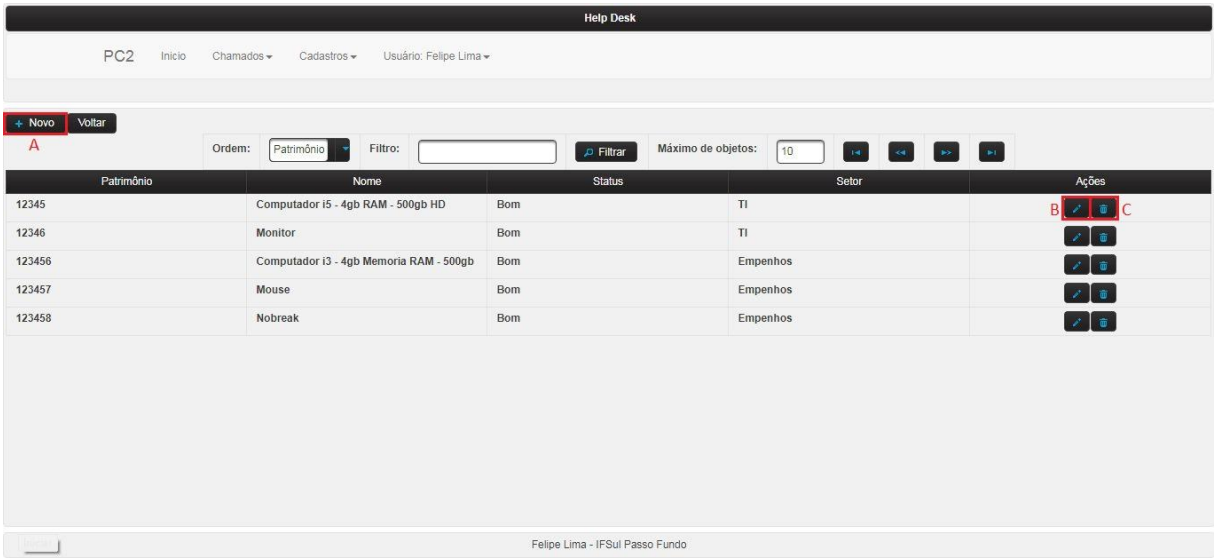
Figura 26 – Fragmento de tela apresentando o menu de cadastros, disponíveis ao perfil de "Administrador".



Fonte: Do Autor

As telas de cadastro possuem uma listagem dos registros referentes a cada um dos cadastros, que possuem um botão "Novo" (Figura 27-A) que irá direcionar para um formulário onde serão inseridas as informações referentes à opção do cadastro selecionado (conforme seleção no menu). Além disso, há as opções de alterar um registro (Figura 27-B) e excluir um registro (Figura 27-C).

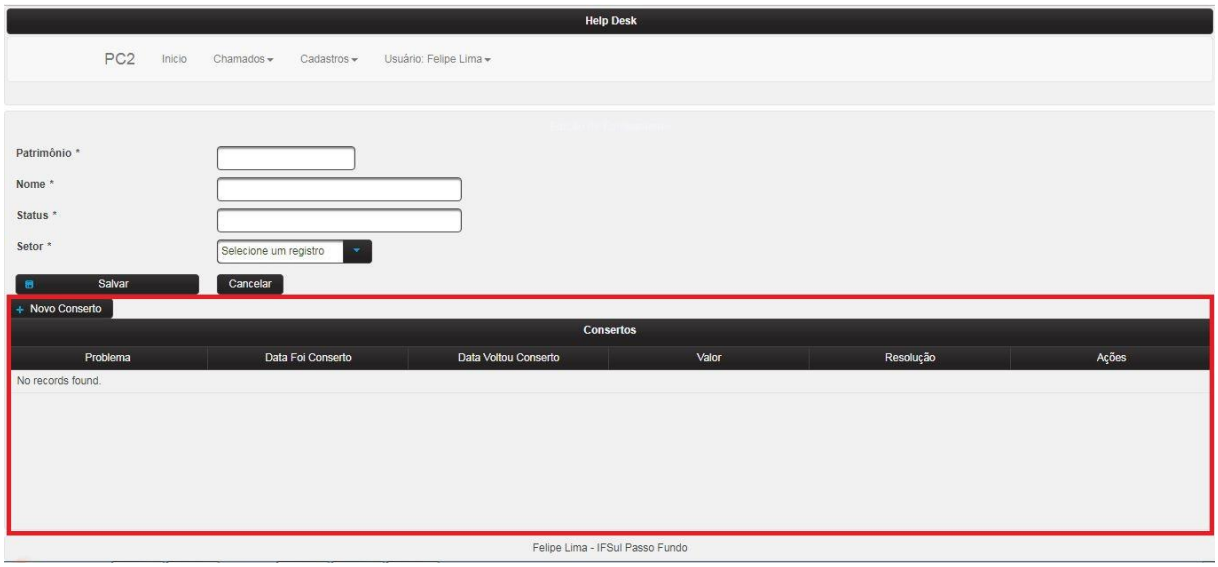
Figura 27 - Tela de listagem das manutenções, no exemplo a tela de equipamentos.



Fonte: Do Autor

Na manutenção de equipamentos, além do formulário para inserção ou alteração de um registro, possui também uma área destinada as informações de consertos desse equipamento (conforme evidenciado na Figura 28). Caso uma peça ou equipamento precise ser enviado para conserto às informações referentes a essa manutenção são inseridas e listadas nessa área. Para registro de um conserto é informado o nome da empresa para a qual foi enviado o equipamento, a data em que foi enviado e a data que retornou do conserto, bem como o valor e o detalhamento do conserto.

Figura 28 – Tela de cadastro de equipamento, contendo seu histórico de consertos.



Fonte: Do Autor

Essa mesma ideia foi utilizada no cadastro de usuário, o qual possui uma área destinada ao cadastro e listagem dos perfis de usuário que o mesmo terá, atribuindo assim sua permissão.

5.5 AVALIAÇÃO DO SISTEMA

Para a realização dos testes, o técnico de TI, com perfil de “Administrador” cadastrou três usuários, cada um com um perfil disponível, para realizar a validação do sistema, em que as funções implementadas se mostraram eficientes para o objetivo inicial deste trabalho, que teve uma aprovação de 100% das funcionalidades. Em próxima etapa, o sistema será implementado em fase de teste com uma parte dos usuários. Após a validação será realizada a implantação do sistema.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento de um sistema Web para resolução de chamados, propondo facilitar ao usuário a solicitação de serviços de suporte na área de TI e melhor controle sobre os chamados e serviços realizados, tendo como estudo de caso a Prefeitura Municipal de Caseiros.

Como dificuldades enfrentadas não foi possível concluir a geração dos relatórios, porém sem prejuízo às principais funcionalidades do sistema.

Então, pode-se concluir que os objetivos propostos foram alcançados, o que contribuiu no aperfeiçoamento do aprendizado das ferramentas utilizadas no desenvolvimento.

Desse modo, como trabalhos futuros verifica-se a necessidade de incluir a geração de relatórios e gráficos, também o aprimoramento das funcionalidades e melhorias no layout da aplicação. Ainda, é possível realizar a criação de novas funções, como o desenvolvimento de um aplicativo para smartphone.

REFERÊNCIAS

BOOCH, G.; RUMBAUGH, J.; JACOBSON I. **UML Guia do Usuário**: Tradução de Fábio Freitas da Silva e Cristina de Amorin Machado. – Rio de Janeiro: Elsevier, 2012. – 12ª reimpressão.

DEITEL, H. M.; DEITEL, P. J. **Java Como Programar**: Tradução da 8ª edição. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

DEITEL, M. Harvey; DEITEL, J. Paul. **Java Como Programar**. 4. ed. Porto Alegre: Bookman, 2002. 1386 p.

GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces**: Tradução da 3ª edição. 3. ed. Rio de Janeiro: Alta Books, 2012.

GONÇALVES, Antônio. **Introdução à Plataforma Java EE 6 com GlassFish 3**. 2. ed. Rio de Janeiro: Ciência Moderna Ltda., 2011.

Help Desk. Disponível em: http://portal.teraware.com.br/sistema-de-help-desk/. Acesso em: 09/12/2018.

HOSTNET. **OsTicket**. Disponível em: <https://www.hostnet.com.br/info/osticket>. Acesso em: 06 dez. 2018.

JPA – O QUE É? PARA QUE SERVE? COMO IMPLEMENTAR UM SISTEMA SIMPLES? 2015. Disponível em: <https://www.dfilitto.com.br/desenvolvimento/java/jpa-o-que-e-para-que-serve-como-implementar-um-sistema/>. Acesso em: 29 maio 2019.

LUCKOW, Décio Heinelmann; MELO, Alexandre Altair. **Programação Java para WEB**. São Paulo: Novatec Editora, 2010.

ORACLE. **Your First Cup.** 2012. Disponível em: <https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html#gjvih>. Acesso em: 24 maio 2019.

OSTICKET - SUPPORT TICKET SYSTEM (Estados Unidos da América (EUA)). **Get The World's Most Popular Customer Support Software.** 2019. Disponível em: <https://osticket.com>. Acesso em: 22 fev. 2019.

Persistência Java – Hibernate e JPA On-Line. Disponível em: <http://www.alfamidia.com.br/ap2/java_hibernate.pdf>. Acesso em 12/12/2018.

TOMTICKET (Brasil). **Help Desk e Atendimento Online na Nuvem**. Disponível em: <https://www.tomticket.com/>. Acesso em: 11 dez. 2018.