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

RICARDO VICENTE HEINZ

SISTEMA INFORMATIZADO PARA SAC DE UMA EMPRESA DE TRANSPORTE DE PASSO FUNDO

Vanessa Lago Machado

PASSO FUNDO
2018

RICARDO VICENTE HEINZ

SISTEMA INFORMATIZADO PARA SAC DE UMA EMPRESA DE TRANSPORTE DE PASSO FUNDO

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

Orientadora: Vanessa Lago Machado

Coorientador: Jorge Luis Boeira Bavaresco

PASSO FUNDO

2018

RICARDO VICENTE HEINZ

SISTEMA INFORMATIZADO PARA SAC DE UMA EMPRESA DE TRANSPORTE DE PASSO FUNDO

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

Banca Examinadora:

_____

Prof.ª. Vanessa Lago Machado

Orientadora

_____

Prof.º. Jorge Luis Boeira Bavaresco

Co-Orientador - Avaliador

_____

Prof.º. Rafael Marisco Bertei

Avaliador

_____

Coordenação do Curso

PASSO FUNDO

2018

À minha esposa e meus filhos,
pela compreensão e o estímulo
em todos os momentos.

À minha mãe e minhas irmãs,
pela minha ausência em alguns momentos.

AGRADECIMENTOS

Agradeço a Deus, pela possibilidade do eterno aprendizado, por todas as dificuldades do caminho, são elas que nos elevam.

Agradeço à minha esposa Ana Cláudia, pela resiliência em aceitar minhas imperfeições e acreditar no esforço do melhoramento contínuo.

Agradeço a todos os professores do curso de Tecnologia em Sistemas para Internet do IFSUL, Campus Passo Fundo, principalmente aqueles que mais me exigiram e cobraram. Um agradecimento especial, a minha orientadora, Vanessa Lago Machado, pelo apoio, paciência e compartilhamento de seu conhecimento.

Por fim, a todos, que de uma forma ou outra tiveram uma parcela de contribuição na construção deste trabalho.

" Você não pode voltar atrás e fazer um novo começo,
mas você pode começar agora e fazer um novo fim. "
Chico Xavier

RESUMO

O presente trabalho expõe os passos necessários para desenvolvimento de um sistema de controle dos chamados abertos por clientes de uma empresa de transporte de passageiros, trazendo uma alternativa de registro ao método atual, que é realizado manualmente. Dessa forma foi implementado um sistema que possibilita a inserção, atualização e exclusão de chamados e movimentos. Para o desenvolvimento da aplicação foi utilizada a tecnologia Java Server Faces em conjunto com as bibliotecas Primefaces e *Hibernate*.

Palavras-chave: *Java Server Faces; Primefaces*; *Hibernate*; SAC.

ABSTRACT

This paper presents the steps required to develop a control system for calls made by customers of a passenger transport company, bringing an alternative to the current method, which is done manually. In this way a system was implemented that enables the insertion, updating and exclusion of calls and movements. For application development, Java Server Faces technology was used in conjunction with the Primefaces and Hibernate libraries.

Keywords: *Java Server Faces; Primefaces*; *Hibernate*; SAC.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Determinada empresa de transporte coletivo de passageiros, na cidade de Passo Fundo, possui um Serviço de Atendimento ao Cliente (SAC), disponibilizado aos seus usuários, para registro das reclamações, elogios, sugestões e denúncias. Atualmente esse serviço é realizado de forma manual, o que demanda mais tempo de execução para realizar a atividade de cada registro, além da dificuldade em buscar informações de datas anteriores, realizar estatísticas dessas informações, como exemplo: quais os motoristas que possuem mais reclamações ou, dentro de um determinado mês, que tipo de reclamação foi mais destacada pelos usuários do sistema de transporte.

Nesse sentido, como tentativa de sistematizar o processo a empresa optou pelo uso do Sistema GLPI, uma aplicação de gestão de serviços e gerenciamento de ativos 100% web, que foi prioritariamente desenvolvida para atender às necessidades de Gestores de TI no gerenciamento de chamados de Helpdesk. Porém, o uso do GLPI não agradou aos profissionais que operam o sistema, por entenderem que sua interface não é amigável, também, por não possibilitar o registro de determinadas informações, como exemplo ônibus, linha, motorista, cobrador, de forma direta, rápida e fácil. Existem no mercado outras plataformas de iteração e atendimento ao cliente, como “Zendesk – *Software* de atendimento ao cliente e sistema de tickets de suporte”, e o “Acelerato – Sistema de SAC Online”, contudo, a demanda recebida pelo SAC da empresa de transporte é baixa, não justificando o investimento em plataformas pagas. Enfim, a empresa deseja um sistema de custo baixo e principalmente customizado.

Foi, então, que surgiu a oportunidade de desenvolver um *software* que possibilite a informatização no processo de atendimento pelo SAC, sendo este o objetivo deste trabalho. Para tanto, o sistema registra a abertura de chamado (reclamações, elogios, sugestões ou denúncias) dos clientes, bem como atualiza os registros de chamados, por parte da atendente e fiscais, e controla o acesso dos usuários.

Nesse sentido o trabalho encontra-se organizado em capítulos, sendo o capítulo 2 uma abordagem do referencial teórico, o qual contém os principais conceitos das tecnologias utilizadas para a construção do sistema. O capítulo 3

aborda a metodologia atual utilizada pela empresa, o sistema proposto e o projeto do novo sistema informatizado, apresentando os diagramas de caso de uso e de classe. O capítulo 4 apresenta o desenvolvimento do sistema, as tecnologias utilizadas com a metodologia MVC. O capítulo 5 mostra o passo a passo de como inserir um novo chamado. Por fim, as considerações finais e as referências.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os principais conceitos utilizados para o desenvolvimento deste trabalho, baseados no estudo das principais literaturas da área.

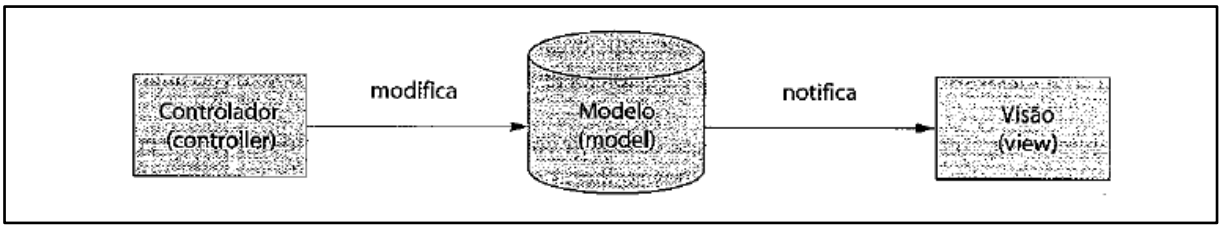
2.1 PADRÃO MVC E DAO

A arquitetura *Model-View-Controller* (MVC – traduzido como Modelo-Visão-Controle) é um conceito importante para desenvolvimento de aplicações Web, pois ela permite separar a aplicação em três partes distintas (camadas), cada qual com sua respectiva funcionalidade. A camada de Modelo está relacionada com o trabalho que aplicação administra, a camada de Visão é responsável por exibir o dado ou informação pertinente à aplicação, e a camada de Controle coordena as partes anteriores, garantindo a exibição da interface correta ou concluindo um determinado trabalho que a aplicação exige (GONÇALVES, 2007).

Gonçalves (2007) segue explicando que, a separação lógica da aplicação nas três partes assegura que cada camada não saiba o que a outra faz. A camada de Modelo representa os dados e não se preocupa de que forma eles são exibidos, pois essa é a função da camada de Visão. Esta por sua vez, não se preocupa de que forma é implementada a lógica do negócio, controlada pela camada de Controle, a qual exerce uma função semelhante a um gerenciador de tráfego, entre as camadas de Modelo e Visão. A camada de Controle direciona as apresentações, com as devidas alterações de dados e recuperações, enviadas pela camada de Modelo.

A Figura 1 ilustra o fluxo da arquitetura MVC, em que por meio do navegador (Visão) são passados parâmetros pela URL para o Controlador. Este por sua vez direciona o fluxo para determinado método de uma *Action*, que utiliza o Bean e o DAO (Controle) para executar a lógica e retorna uma string referente à camada de Visão, que deve ser executada. O Controlador então envia o fluxo para um arquivo de Visão que exibe o HTML no navegador e o fluxo volta ao Controlador onde é encerrado.

Figura 1 - Fluxo padrão da arquitetura MVC



Fonte: DEITEL (2003)

Afirma Gonçalves (2007) que a arquitetura MVC facilita as manutenções nas aplicações, o que agrada grande parte dos desenvolvedores Java, principalmente quem escreve aplicações Web. Uma das principais vantagens ao se utilizar um padrão ao desenvolver aplicações é de poder descrever uma solução genérica para problemas recorrentes.

O padrão *Data Acces Object* (DAO) é o padrão de acesso a dados armazenados em banco de dados mais utilizado (GONÇALVES, 2007). Explica o autor que este padrão disponibiliza uma interface livre, onde pode-se usar para persistir objetos de dados. A ideia é centralizar em um só local, as funcionalidades embutidas no desenvolvimento de acesso e manipulação com os dados, tornando assim, sua manutenção simples. Basicamente o DAO possui métodos que fazem a inclusão, exclusão, listagem e atualização do objeto em um determinado banco de dados.

2.2 JAVA SERVER FACES

A Sun Microsystems apresentou em 2004 um *framework* desenvolvido em conjunto com outras empresas de desenvolvimento, entre elas, IBM, Oracle e Apache, chamado *Java Server Faces (JSF).* A característica deste *framework* era simplificar e facilitar o desenvolvimento de aplicações web, sendo uma evolução do *framework Java Server Pages (JSP),* acrescentou a possibilidade de usar tecnologias modernas com maior facilidade. Também, o JSF adotou o uso da arquitetura MVC, pois utiliza arquivos XML para a construção da visão e classes Java para a lógica da aplicação (JUNEAU, 2013).

JSF é definido como a especificação de *framework* de componentes para se desenvolver aplicações web com linguagem Java. Sua primeira versão apresentou

alguns problemas e em seguida foram lançadas as versões 1.1 e 1.2. Contudo, ainda era acusado de ser muito genérico e que não havia muita aplicação prática, não se tornando interessante e nem aceito pelos desenvolvedores. Com a vantagem de ser altamente flexível e a partir de códigos abertos criados por estes desenvolvedores, a versão 2.0 se mostrou mais simples e fácil de integrar com as demais tecnologias Java, melhorando assim, sua aceitação pelo mercado desenvolvedor (GEARY e HORSTMANN, 2012).

Segundo Gonçalves (2007), o JSF facilita o desenvolvimento das aplicações por meio de componentes de interface de usuário (GUI) conectando esses componentes a objetos de negócio, automatizando o processo de uso de Java Bens e navegação de páginas.

Geary e Horstmann (2012) afirmam que o JSF realiza a interação com os dispositivos clientes fornecendo ferramentas para unir a apresentação visual, as lógicas da apresentação e do negócio de uma aplicação web. Ainda, os autores citam que os serviços mais importantes fornecidos pelo JSF são:

a) **Arquitetura MVC –** as aplicações permitem aos usuários manipularem dados por meio da camada Modelo. O desenvolvedor produz visões ou visualizações do modelo de dados. Sendo em uma aplicação web a tecnologia HTML é utilizada para renderizar a visão. O JSF conecta a camada de visão a de modelo. Enquanto a aplicação JSF, atuando como um controlador, reage às ações do usuário, processando eventos de ação e de mudança de valores, direcionando-os para o código que atualiza o modelo ou a visão.

b) **Conversão de dados –** os dados digitados em formulários web, pelos usuários estão em formato de texto, enquanto o objeto de negócio exige que os dados sejam dos tipos numéricos, datas ou outros. O JSF facilita o trabalho de especificar e customizar as regras para conversão destes dados.

c) **Validação e manipulação de erros –** O JSF facilita a tarefa de criar regras para validar os dados em campos de formulários, como exemplo, um determinado campo ser “obrigatório” ou somente aceitar números. Também, facilitando a tarefa de programar as mensagens de erros aos usuários, quando estes digitam dados inválidos.

d) **Internacionalização –** O JSF gerência aspectos de internacionalização, como a codificação de caracteres e a seleção de *resource bundles*[1].

e) **Componentes customizados –** O JSF possibilita aos desenvolvedores criarem componentes sofisticados que podem ser simplesmente arrastados para dentro das páginas pelos designers.

f) **Suporte Ajax –** O JSF disponibiliza um meio de comunicação Ajax padrão que executa de maneira transparente as chamadas das ações pertinentes ao servidor e atualizações dos componentes do cliente.

g) **Renderizadores alternativos –** O JSF gera, por padrão, *mark-ups*[2] para páginas HTML. Porém, é fácil gerar *mark-ups* para utilizar outras tecnologias de descrição de páginas, como WML ou XUL.

Assim como já foi citado, o JSF possui uma integração com tecnologias, como Ajax, e possui a facilidade de interagir com banco de dados, utilizando JDBC ou EJB. O Java Beans, que é conhecido como JSF Managed Beans, é utilizado na lógica da aplicação oferecendo suporte a conteúdos dinâmicos em todas as visões (JUNEAU, 2013).

O *framework* JSF possui seis fases distintas em seu ciclo de vida, conforme Geary e Horstmann (2012):

1) Restaura Visão (*Restore View*);
2) Aplicar Valores da Requisição (*Apply Request Values*);
3) Validações do Processo (*Process Validations*);
4) Atualização de Valores do Modelo (*Update Model Values*);
5) Chamada da Aplicação (*Invoke Application*);
6) Renderização da Resposta (*Render Response*);

Em conjunto com o JSF, podem ser utilizadas bibliotecas específicas, as quais fornecem recursos adicionais aos padrões existentes na tecnologia do *framework*. Por exemplo, a *Primefaces*, que será apresentada a seguir, pois será uma das tecnologias utilizada no desenvolvimento da aplicação.

---

[1] *Resource Bundle*, nada mais é do que um esquema de arquivos *properties*, onde cada arquivo representa uma língua e cada um desses arquivos possui um conjunto de chaves e valores, sendo que os valores são os textos que serão exibidos na aplicação e estão na língua correspondente à língua que o arquivo representa (SACRAMENTO et al, 2015).

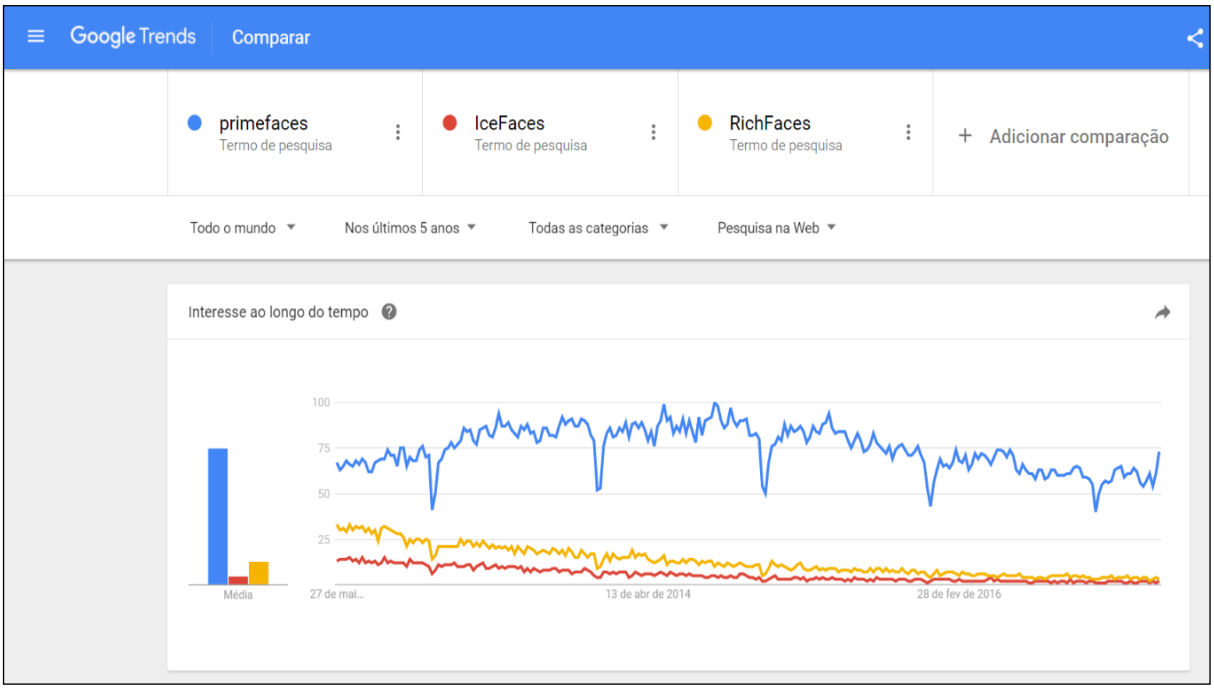
[2] *Mark-ups* são marcações ou *tags*, utilizadas na Renderização das páginas em um navegador ou *browser* web. O HTML – *Hyper-Text Markup Language* é a linguagem de marcação mais utilizada (EIS, 2011).

2.3 PRIMEFACES

O *Primefaces* é um *framework* popular para projetos Java onde se utiliza o JSF. Podemos utilizar o *Primefaces* tanto para desenvolver aplicações sofisticadas para empresas como no desenvolvimento de sites padrão. Tecnicamente, o *Primefaces* é uma biblioteca de componentes de interface gráfica para as aplicações web baseadas em JSF. Com esta biblioteca, facilmente é possível aumentar a produtividade de quem desenvolve, assim como, a experiência do usuário com a aplicação, pois torna menos árduo criar uma aplicação que seja exibida corretamente na maioria dos dispositivos, sem contar com sua flexibilidade, personalização e grande opção de componentes para os mais diversos fins (SCHIECK, 2017).

Primefaces está em alta comparado aos seus concorrentes que estão cada vez mais perdendo espaço. É possível comprovar isso com uma simples busca no Google *Trends (2017)*, como mostra a Figura 2.

Figura 2 - Popularidade dos *frameworks* através do *Google Trends*



Fonte: *GOOGLE TRENDS* (2017).

Para utilizar a biblioteca não é necessário adicionar o componente do *Primefaces* no momento da criação do projeto, nem tanto que esta seja a única

forma de utilizá-lo. Usar o *Primefaces* em um projeto consiste apenas em adicionar sua biblioteca ao projeto, para que suas classes sejam reconhecidas e este processo é extremamente simples para qualquer IDE que se esteja utilizando, basta ter em mãos a biblioteca (.jar) do *Primefaces*. Tal biblioteca pode ser obtida através do endereço: https://www.primefaces.org/downloads/ (SCHIECK, 2015).

2.4 JAVA PERSISTENCE API

Para facilitar o trabalho realizado pelos programadores, principalmente quanto a necessidade de usar códigos de acesso a banco de dados, utilizando queries SQL e melhorar a manutenção de sistemas com códigos mais enxutos, que foi criado o *Java Persistence API* (JPA).

O JPA nada mais é, que um *framework* para persistir dados em Java, que disponibiliza uma API de mapeamento objeto-relacional e soluções para integrar persistência com sistemas corporativos escaláveis. Com este *framework*, basta acrescentar algumas anotações nas classes representativas das entidades no sistema e a partir começar a persistir ou consultar os objetos. Como o JPA não é um produto e sim uma especificação, se faz necessário uma implementação utilizando a biblioteca *Hibernate.* O *Hibernate*, possui módulos, sendo que o *Hibernate* EntityManager é a implementação da JPA que encapsula o *Hibernate Core*, que através de APIs nativas e metadados de mapeamentos em arquivos XML, é a base para o funcionamento da persistência, também utiliza uma linguagem chamada HQL, muito semelhante a SQL, para consultas na base de dados. (FARIAS e JUNIOR, 2015)

3 METODOLOGIA

O presente capítulo apresenta o planejamento para execução do projeto e implementação do protótipo, desde a descrição do método atual, levantamento de requisitos do sistema, desenvolvimento dos diagramas, até a criação do aplicativo.

3.1 MÉTODO ATUAL

A empresa que tem como principal atividade oferecer o serviço de transporte urbano coletivo de passageiros, dessa forma disponibiliza um canal de comunicação aos seus usuários, por meio do SAC, também conhecido como “0800”. Este serviço serve para que o usuário da empresa consulte informações pertinentes às linhas de ônibus, tais como horários, disponibilidade dos ônibus nas linhas, quais linhas atendem determinada região da cidade, entre outras. Mas, também serve como um canal que possibilita ao usuário reclamar ou elogiar determinado serviço ou colaborador, motorista ou cobrador, por exemplo. Além disso, por meio deste canal é possível realizar denúncias, referentes a ações inadequadas e graves que percebam no serviço oferecido.

Atualmente no SAC, os registros de reclamações, elogios ou denúncias, são realizados preenchendo um formulário (Anexos I e II) de forma manual por duas funcionárias que se dividem em turnos de seis horas de trabalho cada, compreendendo o período no dia, das 7 às 19 horas. O serviço funciona durante a semana, de segunda-feira ao sábado. As anotações das informações pertinentes às reclamações, elogios ou denúncias são reproduzidas em um formulário de papel. Simultaneamente, em um arquivo Excel, são anotadas algumas informações, em que posteriormente com uso do próprio Excel, são utilizadas para produzir indicadores estatísticos, como quais os motoristas que possuem mais reclamações ou, dentro de um determinado mês, que tipo de reclamação foi mais destacada pelos clientes. Após finalizar o atendimento ao usuário e o preenchimento do formulário, este é encaminhado ao setor responsável, que tomará ciência e analisará o conteúdo, tomando as medidas necessárias, conforme cada caso, ver Apêndice A.

Neste método manual, algumas dificuldades são observadas: a circulação entre os setores demanda muito tempo, campos obrigatórios podem não serem preenchidos e nem todas as informações registradas, não há registro que permita identificar quem é responsável por cada etapa, buscar informações de registros anteriores, ou seja, de dois mês ou um ano atrás, por exemplo, é uma tarefa onerosa, uma vez que o arquivamento de cada formulário é realizado em arquivos físicos, separados por pastas identificadas pelo ano e mês em que o chamado foi aberto. Outra dificuldade é a obtenção de dados estatísticos ou gráficos comparativos, pois este é um trabalho que deve ser realizado de forma manual, não garantindo a segurança e consistência nos resultados.

3.2 SISTEMA PROPOSTO

No sistema proposto, o usuário (funcionário da empresa), deverá estar logado para operá-lo, informando seu login e senha. A partir daí o sistema oferece um menu, onde é possível escolher a categoria que se quer utilizar. O sistema então, passa a listar os registros referentes a categoria escolhida, possibilitando incluir um novo registro, editar ou excluir um já existente. No caso da categoria “Chamados”, como exemplo, ao clicar no botão “Novo Chamado”, o sistema disponibiliza o formulário para o usuário preencher com as informações inerentes ao chamado aberto e de forma automática a aplicação registrar a data e hora do recebimento do chamado, bem como, o usuário responsável pelo registro, na categoria “Movimentos”. Também será possível ordenar e filtrar os registros exibidos e navegar entre as páginas que exibem a listagem dos mesmos.

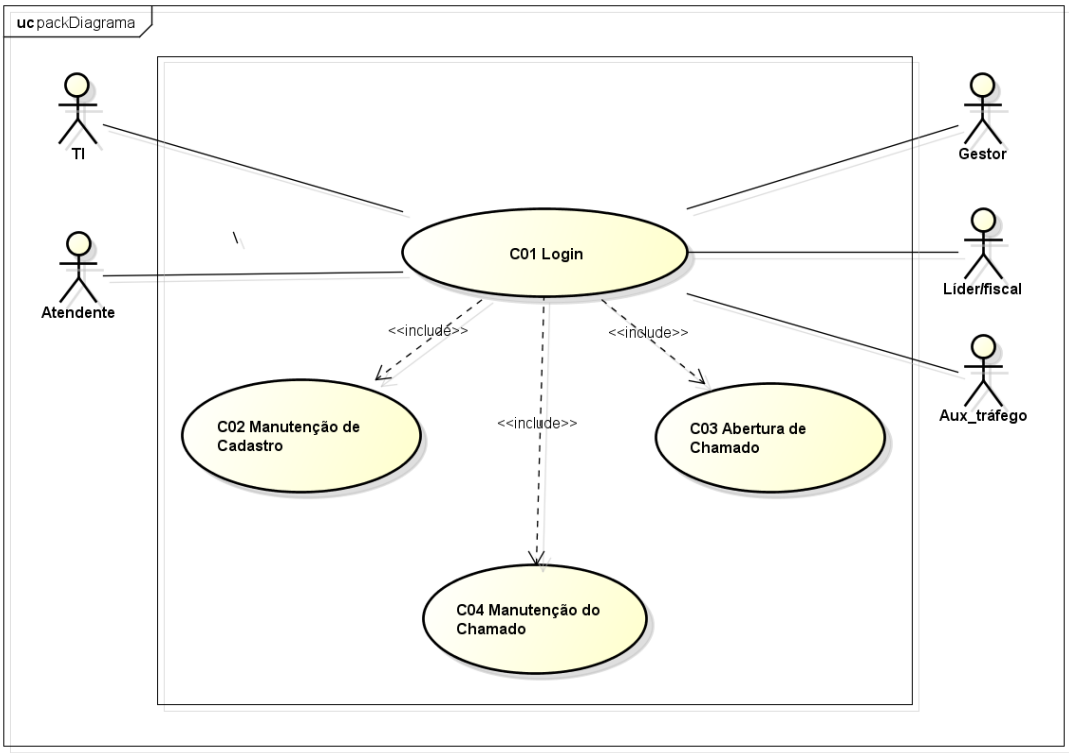
Ainda, por meio do sistema tornar-se-á possível a manutenção dos chamados, fazendo com que o chamado seja aberto, por meio de um atendente, que esse seja encaminhado ao setor responsável, o qual deverá tomar as devidas providências para solucioná-lo e encaminhar ao gestor para conferência do mesmo. Com todos os registros sistematizados o atendente terá a informação de quando o chamado foi encerrado, podendo realizar um retorno ao cliente (quando for o caso).

3.3 PROJETO DO SISTEMA INFORMATIZADO

O objetivo deste trabalho foi desenvolver um protótipo de um *software* para informatizar o processo de registro de reclamações, elogios, sugestões ou denúncias, seu processamento e controles dos procedimentos. O propósito é trazer maior segurança quanto à informação registrada.

A partir da análise dos requisitos do sistema, sendo os requisitos funcionais: cadastrar o registro de um chamado, editar informações de chamados já existentes na base de dados e realizar o controle de acesso dos usuários, e como requisitos não funcionais: ter uma interface amigável. Assim, foi elaborado o diagrama de casos de uso, em que a Figura 3 ilustra as funcionalidades que o sistema contém e os atores que interagem com ele.

Figura 3 - Diagrama Caso de Uso do Sistema SAC para empresa de transporte



Fonte: DO AUTOR.

A partir da análise realizada foi possível definir os acessos de cada perfil, em que o colaborador do setor da TI será responsável pela manutenção dos cadastros - criar, atualizar e excluir, inclusive o cadastro dos demais usuários, ou seja, Atendente, Auxiliar de Tráfego, Líderes, Fiscais e Gestores. Possui, ainda, o

cadastro de tipo do chamado (reclamações, sugestões, elogios e denúncias), do tipo de reclamações e das linhas de ônibus.

Com os cadastros criados, o sistema permitirá ao Atendente realizar o acesso e registrar um chamado com todas as suas informações. Estarão disponíveis ao Atendente as funcionalidades de criar um novo chamado, editar ou excluir. Após a criação de um novo chamado, o sistema permitirá o acesso ao chamado para os demais usuários que fazem parte do processo, ou seja, o Auxiliar de Tráfego, Fiscais, Líderes e Gestores. Pessoas com perfis de atendente e auxiliar de tráfego terão as mesmas funcionalidades disponíveis no sistema. Já para os demais usuários será possível somente editar.

A seguir serão descritos os casos de uso considerados os casos mais complexos em todo processo. A Tabela 1 descreve o caso de uso “C03 - Abertura de Chamado” e a Tabela 2 o caso “C04 - Manutenção do Chamado”, para que se possa ter um melhor entendimento.

Tabela 1 - Descrição de Caso de Uso C03

ITEM	DESCRIÇÃO
Caso de Uso	C03 – Abertura de Chamado
Objetivo	Este caso de uso tem como objetivo abrir um chamado, registrando o tipo de chamado e as informações.
Ator	Atendente e Auxiliar de Tráfego.
Pré-condições	Ter realizado login.
Pós-condições	Sistema apresentar na listagem o chamado salvo.
Fluxo Principal	1) Sistema exibe a listagem dos chamados já cadastrados. 2) Ator seleciona a opção de novo registro. 3) Sistema abre o formulário para inserção dos dados. 4) Ator registra os dados referente ao chamado. 5) Ator seleciona a opção gravar. 6) Sistema grava o novo chamado, exibindo mensagem que os dados foram persistidos com sucesso. 7) Caso de uso encerrado.
Fluxo Secundário	1) Sistema exibe a listagem dos chamados já cadastrados. 2) Ator seleciona a opção de editar registro.

	3) Ator altera os dados necessários do chamado. 4) Ator seleciona a opção gravar. 5) Sistema grava a alteração do chamado, exibindo mensagem que os dados foram persistidos com sucesso. 6) Caso de uso encerrado.

Fonte: do autor.

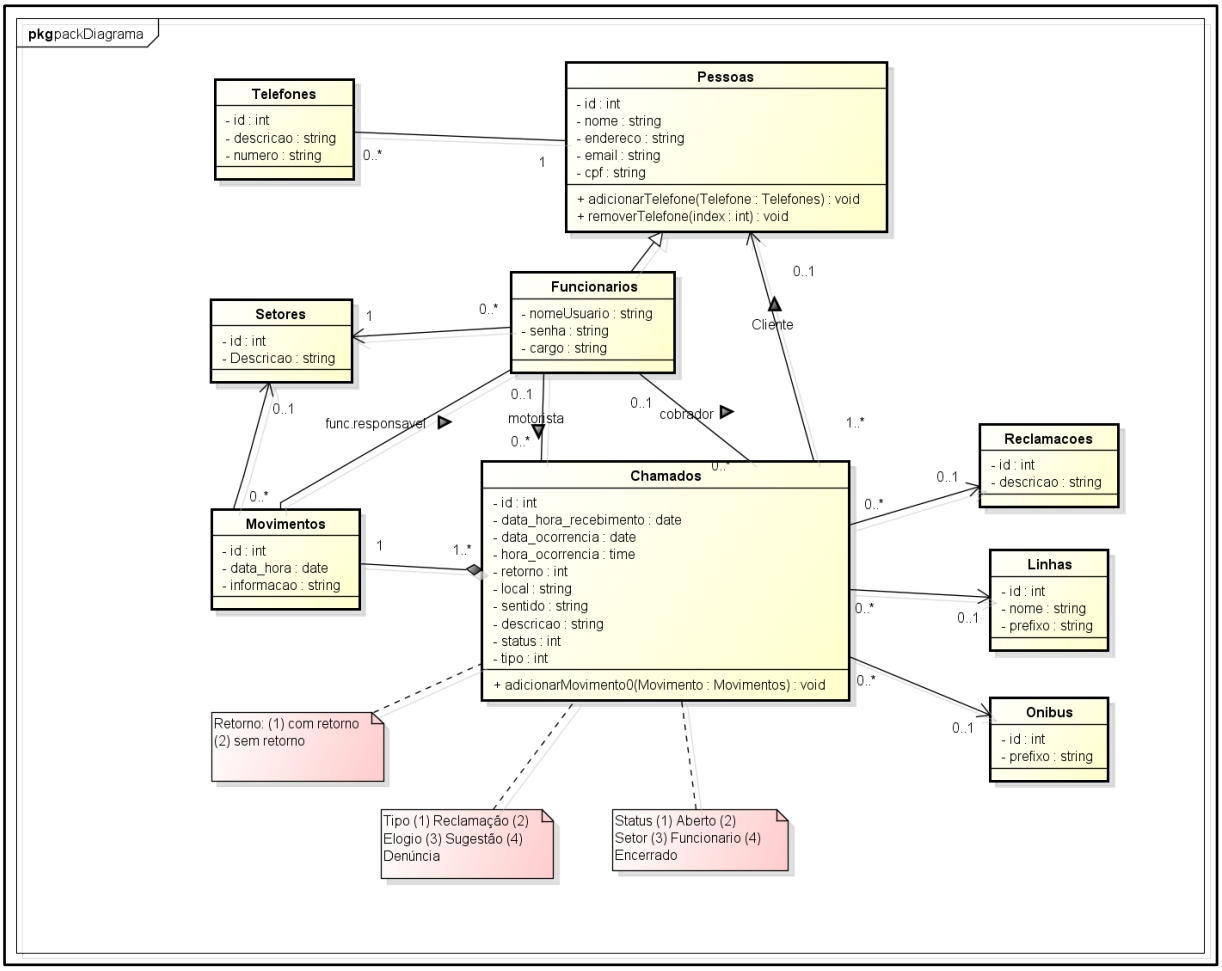
Tabela 2 - Descrição de Caso de Uso C04

ITEM	DESCRIÇÃO
Caso de Uso	C04 – Manutenção do Chamado
Objetivo	Este caso de uso tem como objetivo dar sequência ao atendimento de abertura de um chamado, registrando as informações que foram investigadas.
Ator	Auxiliar de Tráfego, Líder, Fiscal e Gestor.
Pré-condições	Ter realizado login e existir um chamado aberto.
Pós-condições	Sistema lista na aba movimentos o novo registro efetuado.
Fluxo Principal	1) Sistema exibe a listagem dos chamados já cadastrados. 2) Ator seleciona a opção de editar registro. 3) Sistema exibe página do formulário recuperando os dados já gravados em banco de dados. 3) Ator registra os dados referente ao chamado em campo específico. 4) Ator seleciona a opção gravar. 5) Sistema grava as alterações registrada, exibindo a mensagem que os dados foram persistidos com sucesso. 6) Caso de uso encerrado.

Fonte: do autor

Dando sequência à modelagem foi construído o Diagrama de Classes (Figura 4), que tem por principal finalidade mostrar de uma forma sistêmica, as classes ou entidades que estarão envolvidas no projeto, assim como, os atributos de cada uma delas e os relacionamentos entre si.

Figura 4 - Diagrama de Classe do sistema SAC para empresa de transporte



Fonte: DO AUTOR.

A Figura 4 demonstra o relacionamento entre as classes do sistema, sendo que o sistema terá como principal a classe “Chamados”, tendo como atributos: id, data e hora do recebimento do chamado, que é registrada automaticamente por meio da aplicação, data e hora da ocorrência, se há necessidade de retorno ao usuário, local de ocorrência, sentido do ônibus, descrição do chamado, status e tipo, além das vinculações referenciando qual o ônibus da ocorrência, linha, motorista, cobrador e quem é o cliente, caso queira se identificar, e por fim o histórico das tramitações (movimentos) do chamado.

Em cada movimento do chamado será registrada a informação do funcionário responsável pela informação, data e hora do registro e a qual chamado pertence, mantendo assim um histórico.

Ainda, para evitar redundância de dados foi utilizado herança para dados comuns de Pessoas, no qual Funcionários herdam essas informações.

4 DESENVOLVIMENTO DO SISTEMA

Este capítulo aborda o desenvolvimento do sistema, apresentando as tecnologias utilizadas, a estratégia de divisão e organização do código em dois projetos, contendo as camadas referentes à arquitetura MVC e controle de acesso.

4.1 TECNOLOGIAS UTILIZADAS

Para desenvolvimento do sistema foi utilizado à linguagem de programação *Java* (JEE), versão do JDK jdk1.8.0_25, em que foi utilizado o *framework* JSF (*javax.faces*-2.2.14), *Primefaces* 6.1, *Hibernate* 4.2.21, JPA 2.0 e *Hibernate Validator* 5.2.4. Para o Sistema de Gerenciamento de Banco de Dados (SGBD) foi utilizado o *PostgreSQL* 10. Como interface de desenvolvimento do sistema foi utilizada a IDE *NetBeans* 8.2.

4.2 DESENVOLVIMENTO

A ferramenta de administração do SGBD - pgAdmin III, foi utilizada para criação do banco de dados e toda a criação das tabelas, com seus atributos, foi realizada pela aplicação, a partir do *framework Hibernate* interagindo com o JPA. Na Figura 5 é possível visualizar o arquivo “.xml” da unidade de persistência, no qual são inseridas as configurações necessárias para a aplicação efetuar a conexão com o banco de dados. Na *tag* ”*persistence-unit*” é informado o nome do banco de dados e tipo de transação, que no presente projeto foi local. Outras configurações importantes foram informadas com uso da *tag* “*property*”, tais como a url, a porta de comunicação, o nome da aplicação, o drive do *postgresql*, o usuário e a senha do banco de dados, entre outras informações.

Figura 5 - Unidade de persistência

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" 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_2_0.xsd">
  <persistence-unit name="SAC-PC2" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5433/SAC-PC2"/>
      <property name="javax.persistence.jdbc.password" value="******"/>
      <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"/>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Fonte: DO AUTOR.

O *EntityManager* é uma classe que centraliza todas as ações de persistência no banco de dados, em que neste trabalho foi utilizada na classe "*EntityManagerUtil*", conforme Figura 6.

Figura 6 - Classe *EntityManagerUtil*.

```java
package br.edu.ifsul.jpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author Ricardo
 */
public class EntityManagerUtil {

    private static EntityManagerFactory factory = null;
    private static EntityManager em = null;

    public static EntityManager getEntityManager(){
        if(factory == null){
            factory = Persistence.createEntityManagerFactory("SAC-PC2");
        }
        if(em == null){
            em = factory.createEntityManager();
        }
        return em;
    }
}
```

Fonte: DO AUTOR.

Visando o reaproveitamento do código, foram criados dois projetos separados, sendo que no primeiro chamado “SAC-PC2-Persistencia”, (Figura 7 (a)), estão concentrados os arquivos da camada modelo do MVC, e o segundo, chamado “SAC-PC2-Web” (Figura 7 (b)), contempla os arquivos da camada de controle e visão. Na Figura 7 é possível visualizar a organização da árvore de hierarquia dos arquivos dentro de cada projeto. A separação em pacotes facilita o entendimento do projeto, além de acompanhar as melhores práticas e padrões de construção de *software*.

Figura 7 - Projetos SAC-PC2-Persistencia e SAC-PC2, utilizados para desenvolvimento do sistema.

Fonte: DO AUTOR.

A seguir será detalhado como foi o desenvolvimento de cada camada.

4.2.1 Camada de Modelo

A classe “Chamado”, ilustrada na Figura 8, foi criada implementando a interface *Serializable,* a qual permite que a instância do objeto seja transformada em uma sequência de *bytes*, o que é útil quando há necessidade de enviar os objetos pela rede, salvar no disco, ou comunicar de uma JVM com outra, isso porque o estado atual do objeto é mantido de uma ponta a outra sem perder nenhuma informação. A anotação *@Id* é responsável pelo mapeamento da chave primária. A anotação *@SequenceGenerator* define o uso de uma sequência auto incrementável no banco de dados. A anotação *@GeneratedValue* permite o uso dessa sequência para definição automática para o valor do código identificador. Nos demais atributos da classe, a anotação *@Column* especifica como cada coluna será tratada no banco de dados, sendo que o atributo *name* define o nome de cada coluna mapeada na tabela. Nos atributos da classe que manipulam data e hora foi utilizada a anotação *@Temporal,* sendo o atributo da anotação tipo *TemporalType.DATE e TemporalType.TIMESTAMP*. No atributo “descrição”, em sua anotação *@Column*, foi utilizado o atributo *nullable=false* para indicar que a coluna não poderá conter valores nulos, já as anotações *@NotNull(message = "*A descrição deve ser informada*")* e *@NotBlank(message = "*A descrição não pode ser em branco*")* foram utilizadas para informar ao usuário final que no campo do formulário deverá ser digitado alguma informação não podendo deixa-lo em branco, respectivamente.

Também, foram mapeados os relacionamentos que a classe “Chamado” possui com as demais classes que compõe o projeto, em que foi utilizado a anotação *@ManyToOne*, estabelecendo o campo de chave estrangeira por meio das anotações *@JoinColumn* e *@ForeignKey,* em que as relações são “muitos para um”. Já no caso em que a relação é de “um para muitos”, exemplo da relação entre as classes Chamado e Movimento, foi mapeado utilizando uma lista (*List<Movimento>*), obtendo assim uma lista de movimentos dentro de um mesmo chamado, sendo que a anotação utilizada foi *@OneToMany*.

Nas demais classes utilizadas no projeto foram seguidas as mesmas estratégias de codificação para persistir os dados nas tabelas da base de dados.

Figura 8 – Camada modelo - Classe Chamado do projeto SAC-PC2-Persistencia

```java
@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;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "data_hora_recebimento")
    private Calendar data_hora_recebimento;

    @Temporal(TemporalType.DATE)
    @Column(name = "data_ocorrencia")
    private Calendar data_ocorrencia;

    @Column(name = "hora_ocorrencia", columnDefinition = "time")
    private Calendar hora_ocorrencia;

    @Column(name = "retorno")
    private Integer retorno;//2 => sem retorno   1 => com retono

    @Column(name = "local", length = 50)
    private String local;

    @Column(name = "sentido")
    private String sentido;

    @NotNull(message = "A descrição deve ser informada")
    @NotBlank(message = "A descrição não pode ser em branco")
    @Column(name = "descricao", nullable = false, columnDefinition = "text")
    private String descricao;

    @Column(name = "status")
    private Integer status;//1-aberto 2-setor 3-funcionario 4-fechado

    @Column(name = "tipo")
    private Integer tipo;//1-reclamação 2-elogio 3-sugestão 4-denuncia

    @ManyToOne
    @JoinColumn(name = "reclamacao", referencedColumnName = "id")
    @ForeignKey(name = "fk_reclamacao_id")
    private Reclamacao reclamacao;

    @ManyToOne
    @JoinColumn(name = "usuario", referencedColumnName = "id")
    @ForeignKey(name = "fK_usuario_id")
    private Usuario usuario;

    @ManyToOne
    @JoinColumn(name = "linha", referencedColumnName = "id")
    @ForeignKey(name = "fk_linha_id")
    private Linha linha;

    @ManyToOne
    @JoinColumn(name = "onibus", referencedColumnName = "id")
    @ForeignKey(name = "fk_onibus_id")
    private Onibus onibus;

    @ManyToOne
    @JoinColumn(name = "motorista", referencedColumnName = "id")
    @ForeignKey(name = "fk_funcionario_id")
    private Funcionario motorista;

    @ManyToOne
    @JoinColumn(name = "cobrador", referencedColumnName = "id")
    @ForeignKey(name = "fk_funcionario_id")
    private Funcionario cobrador;

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

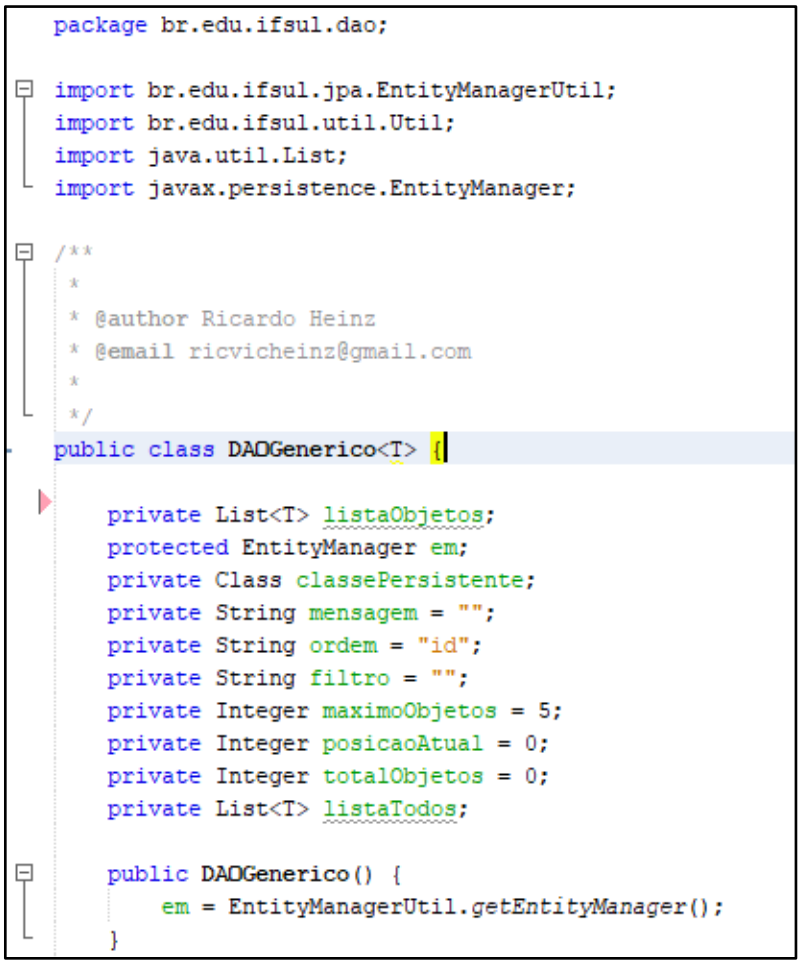
Fonte: DO AUTOR.

Em todas as classes também foram criados os métodos: construtor, *getters* e *setters.* No caso da classe “Chamado”, no método construtor foi definido o valor para os atributos: “data_hora_recebimento”, sendo atribuído a data e hora do sistema; e “status”, definindo o valor em “1”, definindo assim o valor de “aberto” ao status do chamado.

4.2.2 Camada de Controle

No Projeto “SAC-PC2-Web” é responsável pelas camadas de Controle e Visão. A camada de Controle está dividida nos pacotes: Controle, *Converters* e DAO, em que no pacote DAO foi criado uma classe genérica chamada “DAOGenerico”, representada na Figura 9. Essa classe é responsável pelos métodos comuns a todas as classes do projeto, além disso, ela possui os métodos que controlam a paginação, ordenação e os filtros das páginas de listagens e os métodos que realizam as ações na base de dados.

Figura 9 -– Camada Controle - Classe DAOGenerico



```
package br.edu.ifsul.dao;

import br.edu.ifsul.jpa.EntityManagerUtil;
import br.edu.ifsul.util.Util;
import java.util.List;
import javax.persistence.EntityManager;

/**
 *
 * @author Ricardo Heinz
 * @email ricvicheinz@gmail.com
 *
 */
public class DAOGenerico<T> {

    private List<T> listaObjetos;
    protected EntityManager em;
    private Class classePersistente;
    private String mensagem = "";
    private String ordem = "id";
    private String filtro = "";
    private Integer maximoObjetos = 5;
    private Integer posicaoAtual = 0;
    private Integer totalObjetos = 0;
    private List<T> listaTodos;

    public DAOGenerico() {
        em = EntityManagerUtil.getEntityManager();
    }
```

Fonte: DO AUTOR.

Desse modo, todas as classes DAO herdam da classe “GenericoDAO”, reaproveitando o código das implementações, como ilustrada na Figura 10.

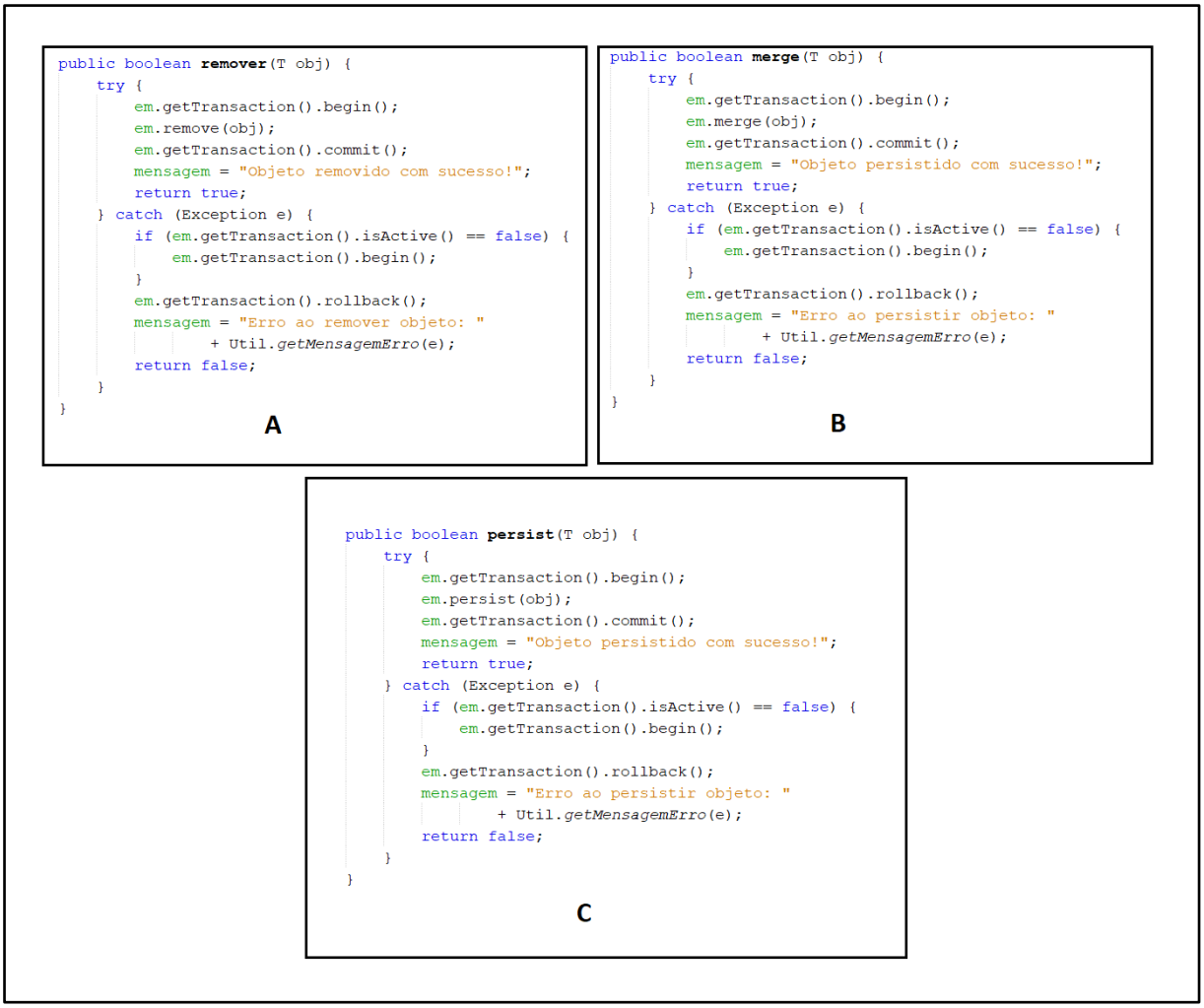
Figura 10 – Camada Controle - Classe ChamadoDAO

```java
package br.edu.ifsul.dao;

import br.edu.ifsul.modelo.Chamado;
import java.io.Serializable;

/**
 *
 * @author Ricardo Heinz
 * @email ricvicheinz@gmail.com
 *
 */
public class ChamadoDAO<T> extends DAOGenerico<Chamado> implements Serializable {

    public ChamadoDAO(){
        super();
        super.setClassePersistente(Chamado.class);
        super.setOrdem("id");
    }

}
```

Fonte: DO AUTOR.

Ainda, em relação a classe “GenericoDAO”, algumas ações de relação direta com o banco de dados podem ser vistas na Figura 11, em que a Figura 11 (a) ilustra o método responsável por remover o objeto da base de dados, a Figura 11 (b) o método responsável por atualizar os dados e a Figura 11 (c) o método responsável por salvar o objeto no banco de dados.

Figura 11 – Camada Controle - Métodos remover, persist e merge da Classe DAOGenerico



A

```
public boolean remover(T obj) {
    try {
        em.getTransaction().begin();
        em.remove(obj);
        em.getTransaction().commit();
        mensagem = "Objeto removido com sucesso!";
        return true;
    } catch (Exception e) {
        if (em.getTransaction().isActive() == false) {
            em.getTransaction().begin();
        }
        em.getTransaction().rollback();
        mensagem = "Erro ao remover objeto: "
                + Util.getMensagemErro(e);
        return false;
    }
}
```

B

```
public boolean merge(T obj) {
    try {
        em.getTransaction().begin();
        em.merge(obj);
        em.getTransaction().commit();
        mensagem = "Objeto persistido com sucesso!";
        return true;
    } catch (Exception e) {
        if (em.getTransaction().isActive() == false) {
            em.getTransaction().begin();
        }
        em.getTransaction().rollback();
        mensagem = "Erro ao persistir objeto: "
                + Util.getMensagemErro(e);
        return false;
    }
}
```

C

```
public boolean persist(T obj) {
    try {
        em.getTransaction().begin();
        em.persist(obj);
        em.getTransaction().commit();
        mensagem = "Objeto persistido com sucesso!";
        return true;
    } catch (Exception e) {
        if (em.getTransaction().isActive() == false) {
            em.getTransaction().begin();
        }
        em.getTransaction().rollback();
        mensagem = "Erro ao persistir objeto: "
                + Util.getMensagemErro(e);
        return false;
    }
}
```

Fonte: DO AUTOR.

No pacote “controle” estão as classes referente a camada de controle, que basicamente são responsáveis por fazer a ligação entre as ações solicitadas pela camada de visão e as regras de negócio contidas na camada de modelo, também as iterações com a base de dados. A Figura 12 ilustra a classe “ControleChamado” com seus atributos e o método construtor. Foi utilizada a anotação *@SessionScoped*, que permite manter instanciada a classe enquanto tiver a sessão ativa. Pela implentação do método construtor da classe é possível instanciar cada objeto que se relaciona com a classe persistente. Temos também, o método “listar” {*return* "/privado/chamado/listar?*faces-redirect=true*";} – responsável por retornar o caminho para montar a URL que irá listar os chamados existente dentro de uma página JSF.

Figura 12 – Camada Controle - Classe ControleChamado

```
@ManagedBean(name = "controleChamado")
@SessionScoped
public class ControleChamado implements Serializable {

    private ChamadoDAO dao;
    private Chamado objeto;
    private FuncionarioDAO daoFuncionario;
    private PessoaDAO daoPessoa;
    private UsuarioDAO daoUsuario;
    private LinhaDAO daoLinha;
    private OnibusDAO daoOnibus;
    private ReclamacaoDAO daoReclamacao;
    private Movimento movimento;
    private Boolean novoMovimento;

    public ControleChamado() {
        dao = new ChamadoDAO();
        daoFuncionario = new FuncionarioDAO();
        daoPessoa = new PessoaDAO();
        daoLinha = new LinhaDAO();
        daoOnibus = new OnibusDAO();
        daoUsuario = new UsuarioDAO();
        daoReclamacao = new ReclamacaoDAO();
    }
```

Fonte: DO AUTOR.

Na Figura 13 é possível visualizar os demais métodos que compõem a classe “ControleChamado”, em que o método “novo” realiza a chamada do formulário para cadastrar um novo objeto. No método “salvar” é realizada uma verificação através do objeto.*getId*()id, sendo que quando esse está nulo, significa que será criado um novo objeto e o mesmo é persistido na base de dados, caso contrário é realizada uma atualização no objeto que já existe na base de dados, informando o id.

O método “editar” (Figura 13) é responsável pela consulta do chamado na base de dados (por meio do método “localizar”). Dessa forma, na Visão é possível popular o formulário com as informações atuais do objeto, permitindo a edição dos valores do chamado correspondente.

Igualmente ao “editar”, o método “remover” (Figura 13) captura o objeto da base de dados por meio do método “localizar”, enviando para o método “remover” da classe “DAOGenerico”, que por sua vez deleta da base de dados o objeto solicitado.

Ainda, na classe “ControleChamado”, existem dois métodos relacionados a classe “Movimento”, que são: “novoMovimento” e “SalvarMovimento” (Figura 13). Assim é possível por meio da Visão, ao consultar determinado chamado, incluir um movimento relacionado.

Figura 13 – Camada Controle - Métodos da classe ControleChamado

```
public String novo() {
    objeto = new Chamado();
    return "formulario";
}

public void salvar() {
    boolean persistiu;
    if (objeto.getId() == null) {
        persistiu = dao.persist(objeto);
    } else {
        persistiu = dao.merge(objeto);
    }
    if (persistiu) {
        Util.mensagemInformacao(dao.getMensagem());
    } else {
        Util.mensagemErro(dao.getMensagem());
    }
}

public void editar(Integer id) {
    objeto = (Chamado) dao.localizar(id);
}

public void remover(Integer id) {
    objeto = (Chamado) dao.localizar(id);
    if (dao.remover(objeto)) {
        Util.mensagemInformacao(dao.getMensagem());
    } else {
        Util.mensagemErro(dao.getMensagem());
    }
}

public void novoMovimento() {
    movimento = new Movimento();
    novoMovimento = true;
}

public void salvarMovimento() {
    if (novoMovimento) {
        objeto.adicionarMovimento(movimento);
    }
    Util.mensagemInformacao("Movimento salvo com sucesso!");
}
```

Fonte: DO AUTOR.

4.2.3 Camada de Visão

A Figura 14 mostra o arquivo “*template*” utilizado para criação do menu, que é inserido em todas as páginas navegadas. Com o uso da *tag* “*ui*:*insert*” os conteúdos das diferentes visões são montados dinamicamente, seja, por exemplo, uma listagem ou um formulário. Desta forma não é necessário que as mudanças sejam realizadas individualmente, em cada página. Para a utilização do *template*, o mesmo foi indicado em todas as páginas na *tag* “*ui:composition*”.

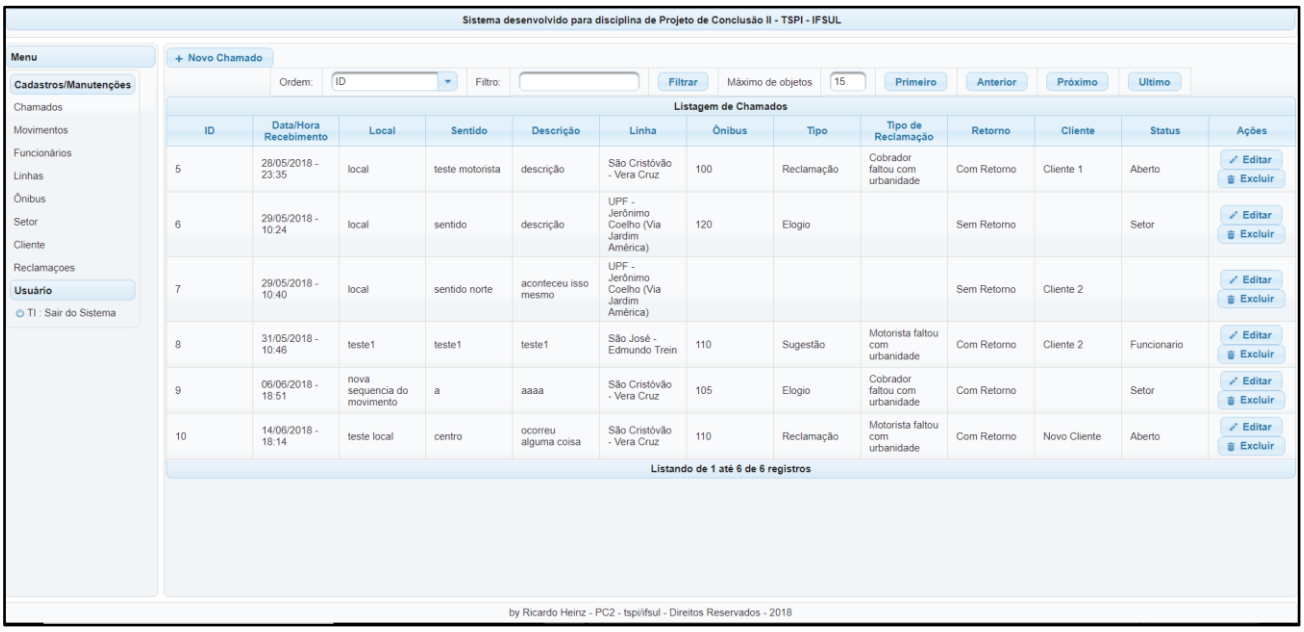
Figura 14 – Conteúdo do arquivo *Template*.xhtml

```
<f:view encoding="ISO-8859-1" contentType="text/html">
    <h:head>
        <title><ui:insert name="titulo">titulo da pagina</ui:insert></title>
        <h:outputStylesheet library="css" name="estilos.css"/>
        <h:outputScript library="js" name="scripts.js"/>
    </h:head>
    <p:layout fullPage="true">
        <p:layoutUnit position="north" header="Sistema desenvolvido para disciplina de Projeto de Conclusão II - TSPI - IFSUL"
                      style="text-align: center">
        </p:layoutUnit>
        <p:layoutUnit position="west" header="Menu">
            <h:form id="menu">
                <p:menu>
                    <p:submenu label="Cadastros/Manutenções" rendered="#{controleLogin.usuarioLogado != null}">
                        <p:menuitem value="Chamados" action="#{controleChamado.listar()}" ajax="false"/>
                        <p:menuitem value="Movimentos" action="#{controleMovimento.listar()}" ajax="false"/>
                        <p:menuitem value="Funcionários" action="#{controleFuncionario.listar()}" ajax="false"/>
                        <p:menuitem value="Linhas" action="#{controleLinha.listar()}" ajax="false"/>
                        <p:menuitem value="Ônibus" action="#{controleOnibus.listar()}" ajax="false"/>
                        <p:menuitem value="Setor" action="#{controleSetor.listar()}" ajax="false"/>
                        <p:menuitem value="Cliente" action="#{controleUsuario.listar()}" ajax="false"/>
                        <p:menuitem value="Reclamaçoes" action="#{controleReclamacao.listar()}" ajax="false"/>
                    </p:submenu>
                    <p:submenu label="Usuário">
                        <p:menuitem value="#{controleLogin.usuarioLogado.nome} : Sair do Sistema"
                                    rendered="#{controleLogin.usuarioLogado != null}"
                                    ajax="false"
                                    action="#{controleLogin.efetuarLogout()}"
                                    icon="ui-icon-power"/>
                        <p:menuitem value="Entrar no Sistema"
                                    rendered="#{controleLogin.usuarioLogado == null}"
                                    ajax="false"
                                    action="#{controleLogin.paginaLogin()}"
                                    icon="ui-icon-person"/>
                    </p:submenu>
                </p:menu>
            </h:form>
        </p:layoutUnit>
        <p:layoutUnit position="center">
            <ui:insert name="conteudo">
            </ui:insert>
        </p:layoutUnit>
        <ui:insert name="dialogos"></ui:insert>
        <p:layoutUnit position="south">
            <ui:include src="/templates/ajaxstatus.xhtml"/>
            <div align="center">by Ricardo Heinz - PC2 - tspi/ifsul - Direitos Reservados - 2018</div>
        </p:layoutUnit>
    </p:layout>
</f:view>
```

Fonte: DO AUTOR.

A Figura 15 apresenta a listagem dos chamados, na qual o usuário pode interagir através de várias ações. Na lateral direita estão localizados os botões de “editar” e “excluir”. Na parte superior é possível navegar entre a paginação da listagem, efetuar ordenação e filtro, também está localizado o botão para incluir um novo chamado, que quando acionado direciona o fluxo à tela o formulário (Figura 16).

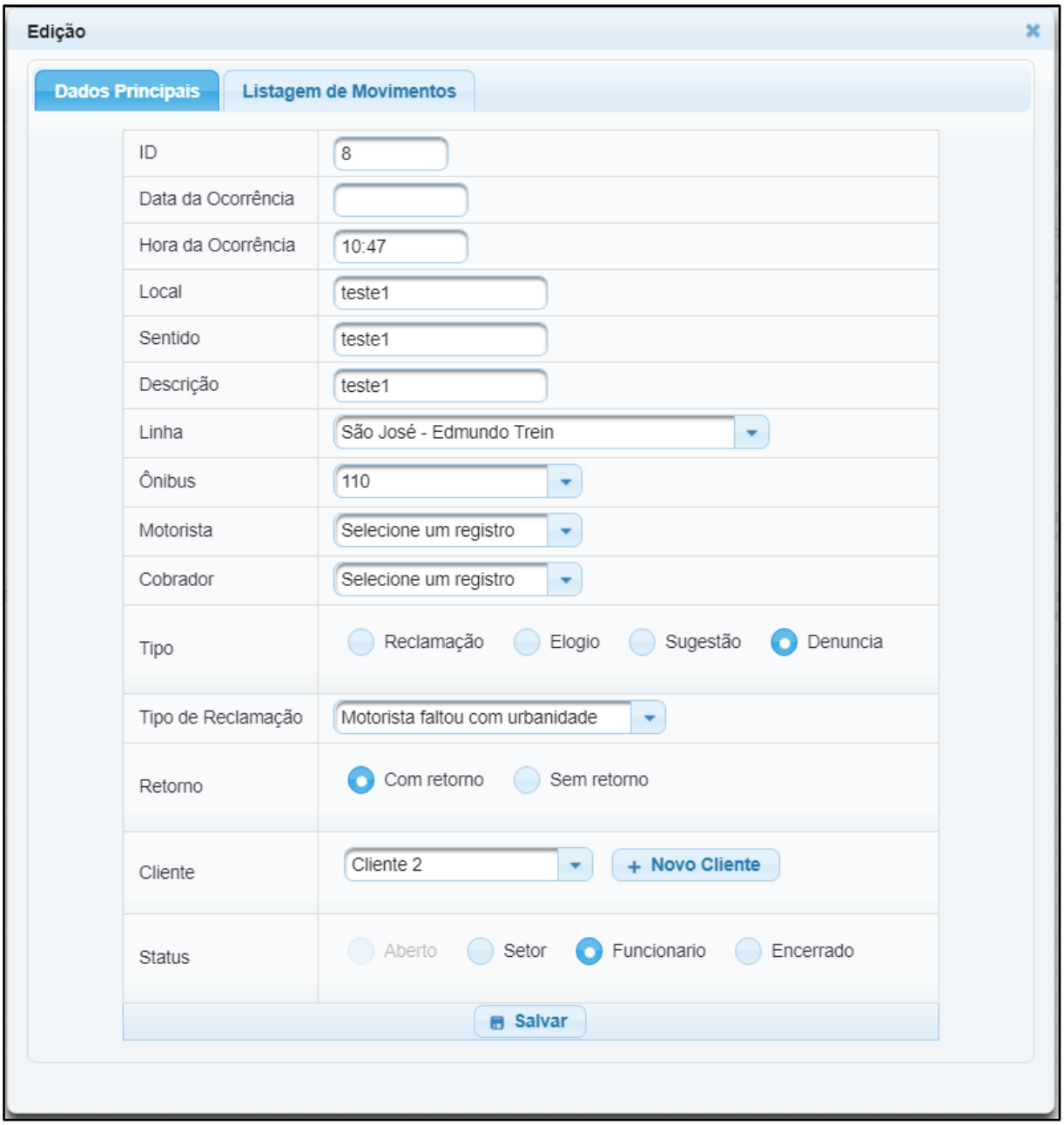
Figura 15 - Listagem de Chamados



Fonte: DO AUTOR.

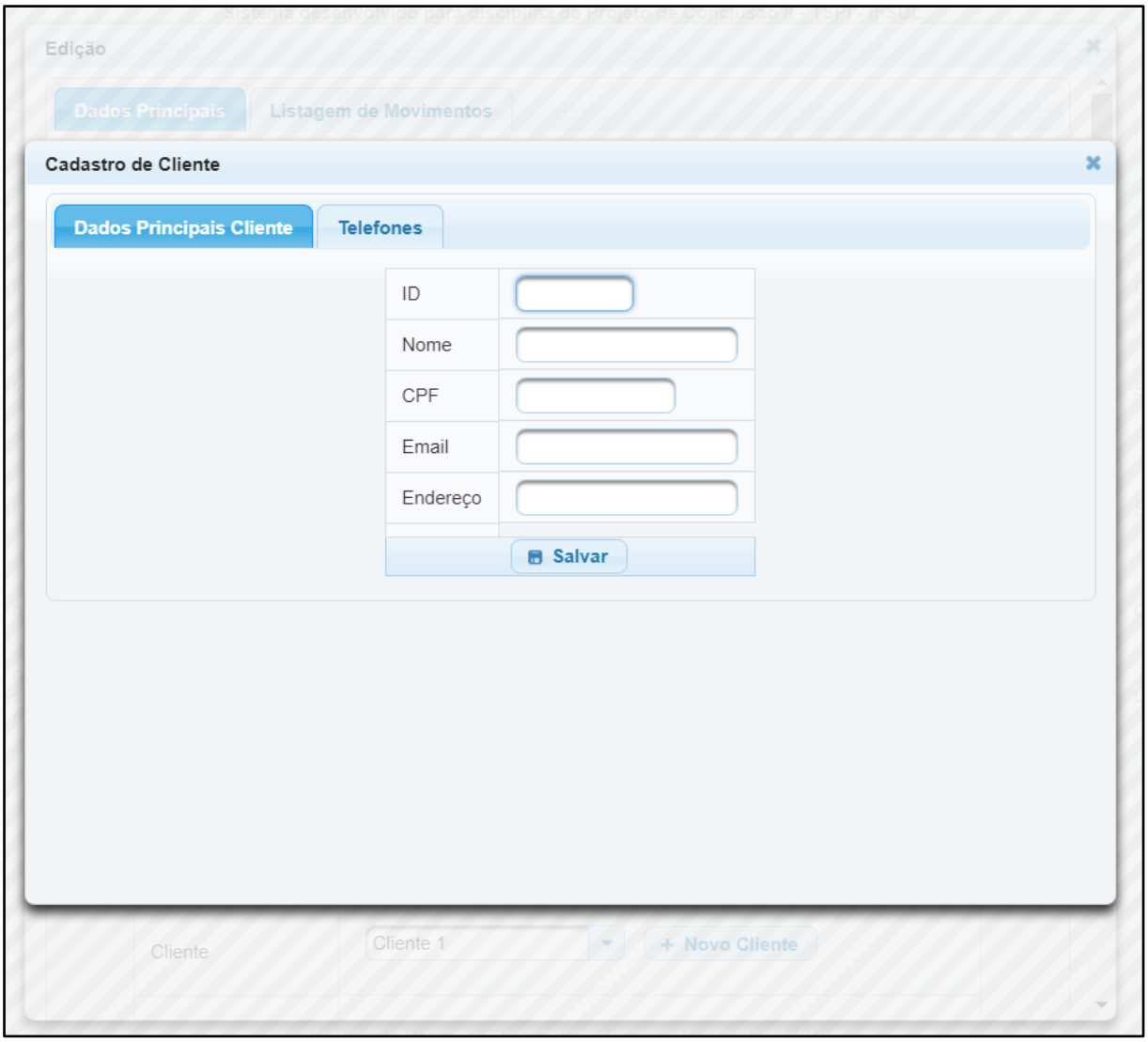
Ao editar um chamado, ilustrado na Figura 16, além de popular as informações inerentes ao chamado (na aba “Dados Principais”), é possível verificar os movimentos relacionados e incluir novos movimentos (aba “Listagem de Movimentos”). Ainda, na aba “Dados Principais”, é possível incluir um novo cliente (Figura 17), possibilitando a interação do usuário em uma única tela. Essa navegação entre as diferentes listagens e formulários faz-se possível utilizando o recurso de diálogos do *PrimeFaces*, por meio da *tag ”p:dialog*”.

Figura 16 - Formulário de edição/ adição de Chamado



Fonte: DO AUTOR.

Figura 17 - Formulário de adição de Cliente pela tela de adição/ edição de chamado.



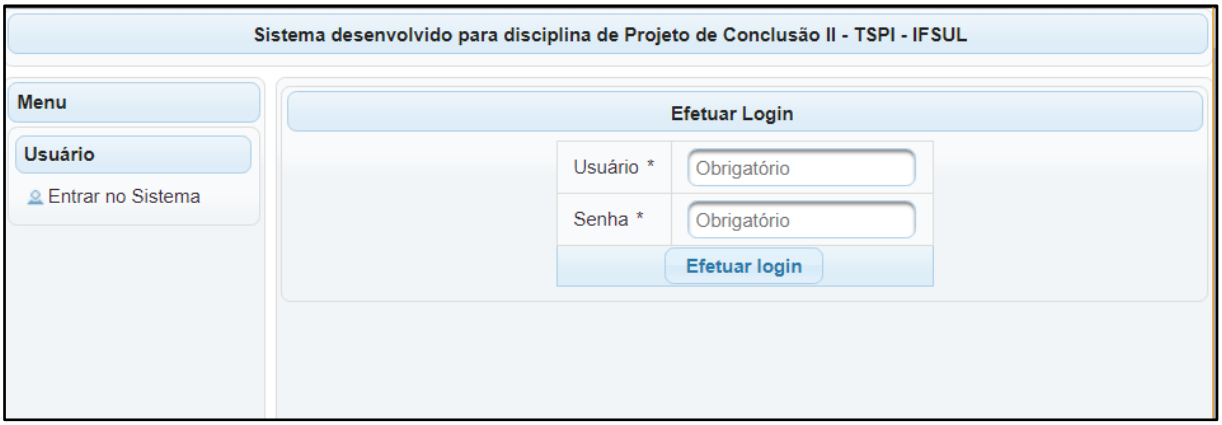
Fonte: DO AUTOR.

As demais visões seguem o mesmo padrão de listagem e formulário.

4.2.4 Controle de Acesso

O controle de acesso ao sistema registra a sessão do usuário, para isso há a necessidade de realizar *login* no sistema (Figura 18), tal controle é realizado no arquivo “*template*”.

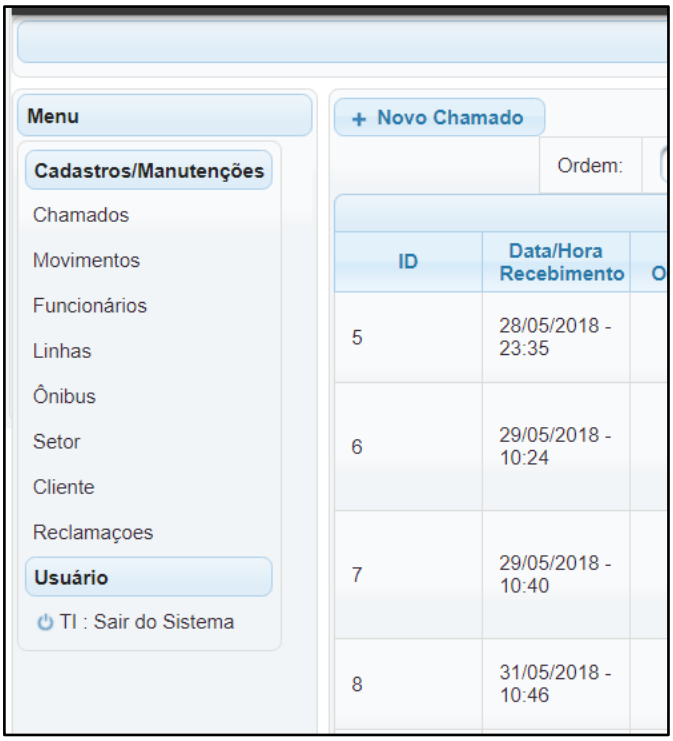
Figura 18 – Controle de acesso do sistema: Formulário de *login*.



Fonte: DO AUTOR.

Nesse sentido o submenu, responsável por disponibilizar as categorias para navegação, só é redenrizado se houver um usuário logado, caso contrário o sistema direciona o fluxo para a página de acesso ao sistema, no qual é apresentado o formulário de *login*. Assim, a Figura 19 apresenta a tela de navegação do usuário logado no sistema, sendo possível visualizar o menu completo para navegação.

Figura 19 – Controle de acesso do sistema: Usuário logado.



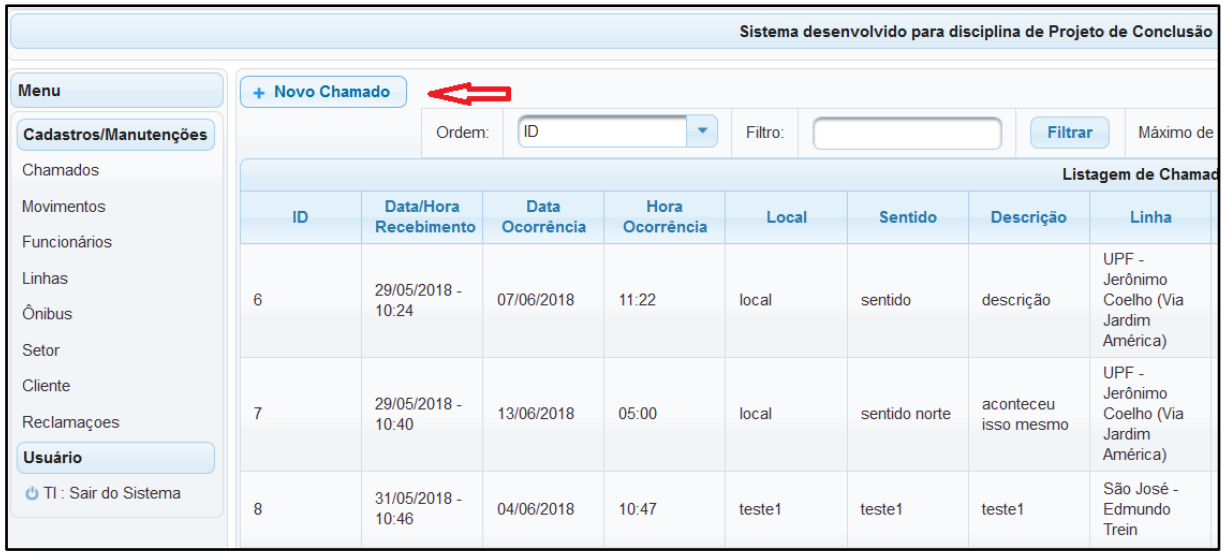
Fonte: DO AUTOR.

5 UTILIZANDO O SISTEMA

Este capítulo descreve de forma simples e visual, através das telas do sistema, o passo a passo de como incluir um novo chamado, considerando que o usuário já esteja logado, etapa explicada na seção4.2.4 (Controle de Acesso).

1. Selecionar no Menu de Cadastros/Manutenções a categoria “Chamados”;
2. Clicar no botão “Novo Chamado”, conforme Figura 20;

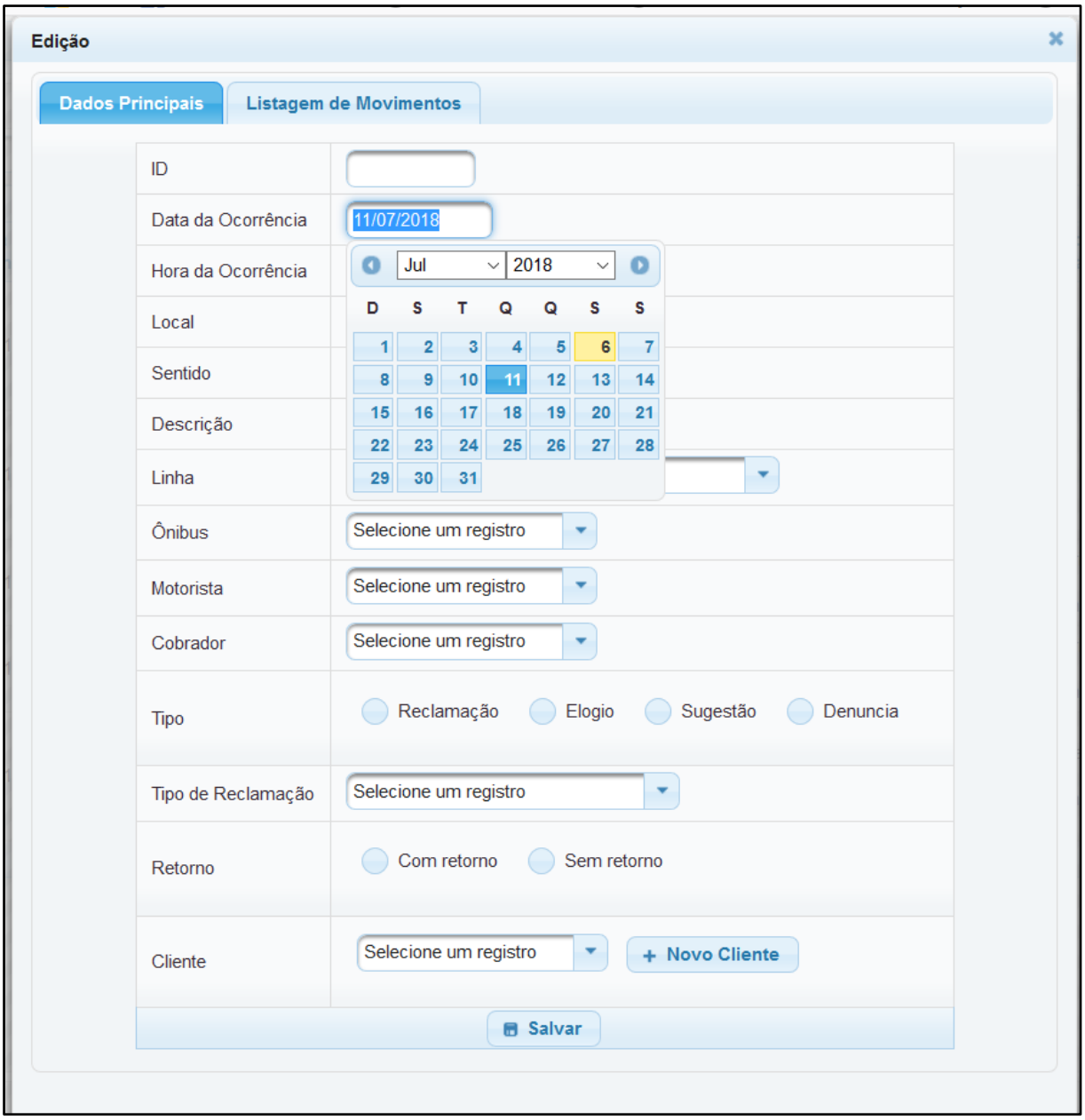
Figura 20 - Selecionar botão Novo Chamado



Fonte: DO AUTOR

3. No formulário aberto, na aba “Dados Principais”, preencher os campos com dados pertinentes ao chamado (Figura 21);
4. Para cadastrar um cliente novo, no caso do cliente solicitar retorno e não ter cadastro, clicar no botão “Novo Cliente” (Figura 21);
5. Preencher os dados referentes ao cliente na aba “Dados Principais Cliente” (Figura 22);

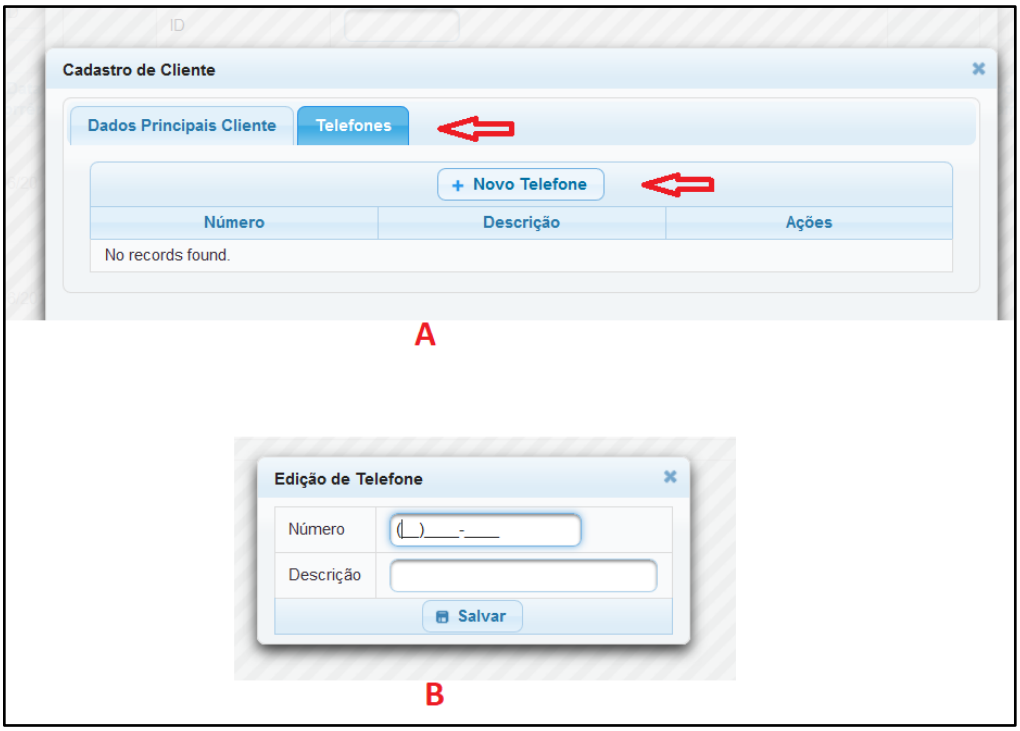
Figura 21 – Formulário de cadastro de chamado, aba “Dados Principais”



Fonte: DO AUTOR

6. Na aba “Telefone” é possível cadastrar números de telefone clicando no botão “Novo Telefone” (Figura 22(a)), informando o número com DDD e uma descrição, conforme Figura 22 (b);

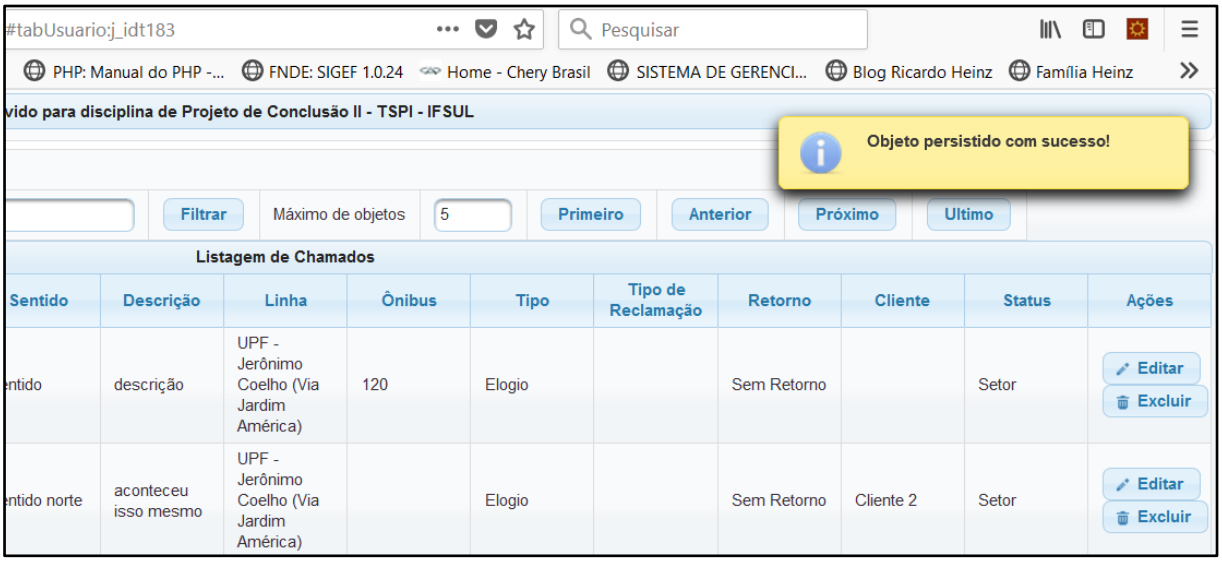
Figura 22 - Cadastrando um novo cliente e seus referidos telefones



Fonte: DO AUTOR

7. Após cadastrar os dados do cliente, selecionar no campo “Cliente” o novo cliente cadastrado;
8. Clicar no botão “Salvar” na aba “Dados Principais”, para salvar as informações do novo Chamado;
9. O sistema informará que o objeto foi persistido com sucesso (Figura 23).

Figura 23 - Objeto do Chamado persistido com sucesso.



Fonte: DO AUTOR

6 CONSIDERAÇÕES FINAIS

O presente trabalho demonstrou os passos necessários para desenvolver um protótipo de um sistema de registro das reclamações, elogios, sugestões e denúncias de uma empresa de transporte de passageiros. Para isso foi escrito o referencial teórico, que possibilitou aprofundar os conhecimentos em várias tecnologias necessárias ao desenvolvimento do protótipo. Assim, com base na análise de requisitos foi desenvolvida a modelagem do sistema.

Em relação as tecnologias utilizadas, como *Java Server Faces*, em conjunto com as bibliotecas do *framework PrimeFaces* e *Hibernate*, mostraram-se eficientes e auxiliaram no desenvolvimento, principalmente na implantação do *layout*, visto que o *framework* possui *templates* para aplicação e implementação de *layouts* de forma simples e com visual agradável.

Os objetivos propostos de registrar a abertura de um chamado (reclamação, elogio, sugestão ou denuncia) do cliente, atualizar os registros do chamado, com a inserção de movimentos e controlar *login* dos usuários, foram atingidos plenamente.

Contudo, no decorrer do desenvolvimento houveram algumas dificuldades enfrentadas, como a alteração das visões (permissão de acesso) de acordo com o perfil do usuário logado.

Dessa forma, como melhoria futura do sistema está à implantação de funcionalidades que possibilitem determinar visões diferentes conforme o usuário que estiver logado, como exemplo, determinado fiscal poder visualizar somente os chamados atribuídos para ele investigar. Também, o desenvolvimento de relatórios que possibilitem ao gestor avaliar a qualidade dos serviços prestados, bem como a geração de gráficos a partir dos dados estatísticos, para auxiliar na visualização dos dados.

REFERÊNCIAS

DEITEL, Harvei M.; DEITEL, Paul J. **Java, como programar.** 4ª Ed. Poarto Alegre, RS, Bookman, 2003.

EIS, Diego. **O básico: O que é o HTML?**. Tableless. 2011. Disponível em: <https://tableless.com.br/o-que-html-basico/>. Acesso em 21 mai. 2017.

FARIA, Thiago; JUNIOR, Normandes. **JPA e *Hibernate***. 1ª Ed. São Paulo, SP, Algaworks, 2015.

GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces**. 3ª Ed., Rio de Janeiro, RJ, Alta Books, 2012.

GOOGLE TRENDS. **Interesse ao longo do tempo.** Disponível em: https://trends.google.com.br/trends/?geo=BR. Acessado em: maio/2017.

GONÇALVES, Edson. **Desenvolvendo Aplicações WEB com: *JSP, SERVLETS, JAVASERVER FACES, HIBERNATE, EJB 3 PERSISTENCE e AJAX***. Rio de Janeiro, RJ, Editora Ciência Moderna, 2007.

JUNEAU, Josh. **Introducing Java EE 7**. 2013. Disponível em: <http://sngo.me/b4THM>. Acesso em: 08 mai. 2017.

NETO, Almir; CALAÇA, Otávio. **Desenvolvimento em Três Camadas com PHP, MVC e Ajax.** SlideShare. 2008. Disponível em: <https://pt.slideshare.net/almirnet/almirnetoeotaviocalacamvcajax122599119955337 68>. Acesso em: 31 mai. 2017.

SACRAMENTO, Cleverson et all. ***Framework* Demoiselle 2.0 – Gia de referência**. DocSlide. 2015. Disponível em: < http://docslide.com.br/documents/demoiselle-reference.html>. Acesso em: 21 mai. 2017.

SCHIECK, Rodrigo. **Introdução ao PrimeFaces.** DevMedia. Disponível em <http://www.devmedia.com.br/introducao-ao-primefaces/33139>. Acesso em: 21 mai. 2017.

APÊNDICE A: Descrição do fluxo de registro de abertura de um chamado

Como exemplo, a seguir é descrito o fluxo de uma reclamação de um usuário que aguardava o ônibus no ponto de embarque/desembarque, solicitou o embarque e o motorista não parou:

1) ***Recebimento da Reclamação –*** o SAC recebe a ligação do usuário, anotando o máximo de informações inerentes a reclamação em formulário específico. Solicitando também ao usuário se o mesmo quer um retorno ou não da reclamação registrada.

2) ***Encaminhamento da Reclamação –*** o SAC encaminha o formulário com as informações anotadas ao setor responsável, no caso, Setor de Tráfego.

3) ***Recebimento pelo Setor –*** uma auxiliar administrativa recebe o formulário, identifica por meio das informações fornecidas pelo usuário, os colaboradores (motorista e cobrador) envolvidos, anotando no formulário em campo específico, após, destina o formulário a um fiscal.

4) ***Processamento –*** o fiscal após receber o formulário, toma ciência do conteúdo da reclamação, entrando em contato com os envolvidos buscando informações do porquê que o motorista não atendeu à solicitação de embarque do usuário. A partir daí, procede com as orientações para que o fato não se repita. Anota no formulário a versão dos colaboradores, suas considerações e medidas adotadas. Devolve o formulário a auxiliar administrativa que encaminha ao gerente do setor. Este por sua vez analisa os dados, observa as medidas adotadas e registra seu parecer.

5) ***Finalização do processo –*** o setor de tráfego devolve o formulário ao SAC. Este por sua vez, realiza a devolutiva ao usuário, caso ele tenha solicitado e arquiva o formulário, fisicamente.

ANEXO I: Folha de registro do chamado (frente)

LOGO	**Reclamação ( ) Elogio( ) Sugestão ( ) Denúncia ( )**

Data Recebimento:	/	/	**Hora:**	:	**Número:** /
Data Ocorrência:	/	/			**Hora Ocorrência:** :

Origem: (T) Telefone (I) E-mail (F) Relatório Fiscal (E) Empresa (S) Site (F) Facebook

Tipo: ( ) Sem retorno ao usuário ( ) Com retorno ao usuário

Nº ônibus:	**Linha:**	**Placa:**	**Sentido:**
Local:			
Relato Usuário:			
Código:	**Responsável atendimento:**		

Setor: ( ) Tráfego ( ) Linhas ( ) Financeiro

Providências do atendimento:					
Data entrega ao Setor:	/	/	**Hora entrega setor:**	:	
Usuário:					
Endereço:					
Bairro:					
Cidade:			**CEP:**		
Telefone:					
E-mail:					

ANEXO II: Folha de registro do chamado (verso)

NO SETOR:

Identificação Motorista:

Identificação Cobrador:

Relato do Funcionário:

Providência do Setor Responsável:

Responsável pelo Setor:

Procedência: ( ) Com Procedência ( ) Sem Procedência

Data Retorno Setor: / / Hora Retorno do Setor: :

Resposta ao Usuário:

Meio: Telefone ( ) ( ) Carta ( ) E-mail

1ª Tentativa: Data	**/ /**	**Hora:**	**:**	**Responsável:**
2ª Tentativa: Data	**/ /**	**Hora:**	**:**	**Responsável:**
3ª Tentativa: Data	**/ /**	**Hora:**	**:**	**Responsável:**

Aceitação Do Usuário: