

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

VINÍCIUS ARAÚJO SILVA

**DELEGACIA VIRTUAL: UM ESTUDO DE CASO DE DESENVOLVIMENTO DE
APLICAÇÃO WEB UTILIZANDO JAVA**

**NATAL/RN
2012**

VINÍCIUS ARAÚJO SILVA

**DELEGACIA VIRTUAL: UM ESTUDO DE CASO DE DESENVOLVIMENTO DE
APLICAÇÃO WEB UTILIZANDO JAVA**

Monografia apresentada ao Curso de Graduação em Bacharelado em Sistemas de Informação do Centro Universitário do Rio Grande do Norte, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Itamir de Moraes Barroca Filho

NATAL/RN

2012

BIBLIOTECA UNI-RN

004.43 A848a - RB: 37268 (M002082)

Título: Delegacia virtual : um estudo de caso ...

Autoria: Silva, Vinícius Araújo

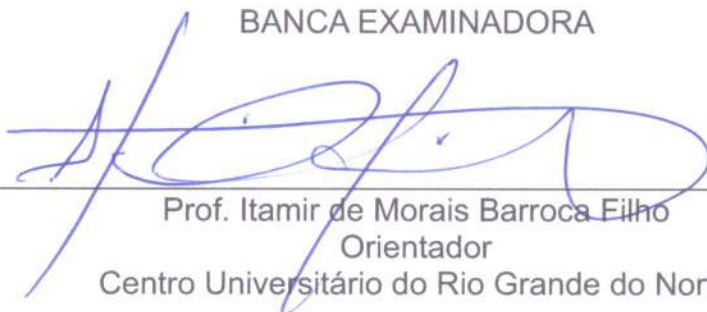
2012, ex. 1

**DELEGACIA VIRTUAL: UM ESTUDO DE CASO DE DESENVOLVIMENTO DE
APLICAÇÃO WEB UTILIZANDO JAVA**

Monografia apresentada ao Curso de Graduação em Bacharelado em Sistemas de Informação do Centro Universitário do Rio Grande do Norte, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em: ____/____/____

BANCA EXAMINADORA

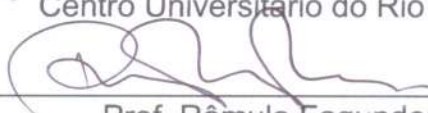


Prof. Itamir de Moraes Barroca Filho
Orientador

Centro Universitário do Rio Grande do Norte



Prof. Ytalo Rosendo do Amaral
Centro Universitário do Rio Grande do Norte



Prof. Rômulo Fagundes Catanhede
Centro Universitário do Rio Grande do Norte

*Dedico este trabalho a meus pais, pelo
apoio incondicional de sempre.*

AGRADECIMENTOS

Agradeço ao meu orientador, prof. Itamir de Moraes Barroca Filho, pela paciência e dedicação com que me auxiliou no desenvolvimento deste trabalho.

Ao Sr. Constantino Emiliano Loiola, meu superior durante o período de estágio na Secretaria de Estado da Segurança Pública e da Defesa Social – SESED do Rio Grande do Norte, pela confiança depositada em mim para realizar o desenvolvimento do projeto da Delegacia Virtual.

RESUMO

Sabe-se que, atualmente, o serviço de emissão de Boletins de Ocorrência no estado do Rio Grande do Norte é de inteira responsabilidade da Polícia Civil. Sabe-se, também, que esse serviço é realizado nas Delegacias de Polícia muitas vezes de forma manual e sem nenhuma padronização. Dessa forma, a necessidade de melhoria desse serviço chamou a atenção da Coordenadoria de Informática e Estatística da Secretaria de Estado da Segurança Pública e da Defesa Social – SESED, que visando disponibilizar um melhor serviço para a população do Rio Grande do Norte, propôs o desenvolvimento de um sistema de informação com o objetivo de atingir todas as Unidades Policiais do estado. O novo sistema de informação, voltado para a área *web*, foi desenvolvido utilizando a linguagem de programação *Java* recebeu o nome de “Delegacia Virtual”. Este trabalho visa, portanto, descrever o desenvolvimento e o funcionamento do sistema de informação, expondo as tecnologias utilizadas em seu processo de criação.

Palavras-chave: Segurança Pública; Boletins de Ocorrência; *Web*; *Java*; Delegacia Virtual.

ABSTRACT

It is known that, nowadays, the Police Report issuing service in the state of Rio Grande do Norte is the responsibility of the Civil Police. It is also known that this service provided at Police Stations is often done manually and with no specific patterns or standardization. Thereby, the need to improve this service caught the attention of the Coordenadoria de Informática e Estatística (Coordinating Body of Informatics and Statistics) of the Secretaria de Estado da Segurança Pública e da Defesa Social – SESED (Estate Bureau of Public Security and Social Protection) who, aiming to provide a better service for the state's population, proposed the development of an information system with the goal of reaching all police units of the state. The new information system, developed for the web, was made using the programming language Java and was named "Delegacia Virtual", or "Virtual Police Station". This paper aims, therefore, to describe how the information system was developed and who it works, exposing the technologies used in its creation process.

Keywords: Public Security; Police Reports; Web; Java; Virtual Police Station.

LISTA DE FIGURAS

Figura 01 – Fluxo do Código-Fonte	21
Figura 02 – Configurando o JPA	22
Figura 03 – JARs para o JPA	23
Figura 04 – Classe Mapeada com JPA	23
Figura 05 – Arquitetura do <i>JavaServer Faces</i>	24
Figura 06 – Configuração da <i>FacesServlet</i>	25
Figura 07 – Padrão de <i>URLs</i>	25
Figura 08 – JARs para o <i>JavaServerFaces</i>	25
Figura 09 – Página com <i>JavaServerFaces</i>	26
Figura 10 – Ciclo de Vida do <i>JavaServerFaces</i>	26
Figura 11 – JARs para o <i>Primefaces</i>	28
Figura 12 – <i>Namespace</i> do <i>Primefaces</i>	28
Figura 13 – Definindo o Tema do <i>Primefaces</i>	28
Figura 14 – JARs para o Tema do <i>Primefaces</i>	29
Figura 15 – <i>Template</i> de Relatório em XML	30
Figura 16 – <i>Template</i> de Relatório no <i>iReport</i>	30
Figura 17 – Código-Fonte para Geração de Relatório <i>JasperReports</i>	31
Figura 18 – Instalação do <i>Apache Tomcat</i> como Serviço	32
Figura 19 – Interface <i>Web</i> para Gerenciar Aplicações	33
Figura 20 - Diagrama de Casos de Uso	34
Figura 21 – Estrutura do Projeto no <i>Eclipse</i>	35
Figura 22 – Classe <i>GenericDao</i>	36
Figura 23 – Entidade do Pacote <i>br.dpvirtual.model</i>	37
Figura 24 – Diagrama de Classes	38
Figura 25 – <i>Managed Bean</i> Responsável pelas Buscas	39
Figura 26 – Declaração de Filtro no <i>web.xml</i>	41
Figura 27 – Conversor Implementado no Sistema	42
Figura 28 – Interface <i>PersistDB</i>	42
Figura 29 – Implementação do <i>LazyListing</i>	44

LISTA DE TABELAS

Tabela 01 – Cronograma de Desenvolvimento do Sistema	32
--	----

SUMÁRIO

1 – INTRODUÇÃO	11
2 – OBJETIVOS	12
3 – SOLUÇÃO PROPOSTA	13
3.1 – Módulo de Boietins	13
3.2 – Módulo de Mandados de Prisão	14
3.3 – Módulo de Impressões	14
3.4 – Módulo de Homologações	15
3.5 – Módulo do CIOSP	16
3.6 – Módulo do DETRAN	17
3.7 – Módulo do ITEP	17
4 – REFERENCIAL TEÓRICO	18
4.1 – <i>Java</i>	18
4.2 – <i>Microsoft SQL Server</i>	19
4.3 – <i>Java API Persistence</i>	20
4.4 – <i>JavaServerFaces</i>	21
4.5 – <i>Primefaces</i>	25
4.6 – <i>JasperReports</i>	27
4.7 – <i>Apache Tomcat</i>	29
5 – ESTUDO DE CASO	32
5.1 – Cronograma de Desenvolvimento do Projeto	32
5.2 – Detalhamento do Código-Fonte	34
5.2.1 – Classes de Persistência (br.dpvirtual.dao)	35
5.2.2 – Classes de Modelo (br.dpvirtual.model & br.dpvirtual.views)	36
5.2.3 – Classes de Controle (br.dpvirtual.controller)	38
5.2.4 – Classes Utilitárias (br.dpvirtual.utils)	39
5.2.5 – Filtros (br.dpvirtual.filter)	40
5.2.6 – Conversores (br.dpvirtual.converter)	41
5.2.7 – Interfaces (br.dpvirtual.interfaces)	42
5.2.8 – APIs (br.dpvirtual.apis)	42
5.2.9 – <i>Webservices</i> (br.dpvirtual.webservices)	43
5.2.10 – <i>Lazy Listing</i> (br.dpvirtual.listing)	43
6 – CONCLUSÃO	45
7 – REFERÊNCIAS BIBLIOGRÁFICAS	46

1 – INTRODUÇÃO

A popularização da *internet* e, conseqüentemente, dos sistemas de informação, tornou o conhecimento um dos bens mais valiosos existentes na atualidade. Por meio desses sistemas, é possível ter tais informações sempre disponíveis, com um acesso rápido e seguro. Convém ressaltar que, atualmente, toda e qualquer instituição, seja ela pública ou privada, precisa que informações necessárias ao seu funcionamento estejam sempre disponíveis, seja para auxiliar na tomada de decisões estratégicas ou para melhorar a prestação de um determinado serviço.

A Polícia Civil é um dos órgãos que compõem a segurança pública do estado e é a responsável pelo serviço de emissão de Boletins de Ocorrência, documentos que registram a notícia do crime e que dão início à fase de apuração do mesmo. No estado do Rio Grande do Norte, a emissão de boletins de ocorrência na maioria das Delegacias de Polícia é feita de forma manual e pouco eficiente, tendo, cada delegacia, seu próprio modelo de boletim, sem nenhum tipo de padronização.

Convém ressaltar que, dessa forma, as informações muitas vezes não são registradas de maneira precisa, e que em alguns casos são encontrados, nos Boletins de Ocorrência, rasuras que comprometem os dados registrados. Além disso, o processo de registro manual facilita a ocorrência da perda do controle da numeração dos boletins, tornando possível que uma numeração seja utilizada mais de uma vez, impossibilitando o fácil acesso a ele quando necessário.

Parece claro afirmar que todos os fatores mencionados acima, juntamente com a necessidade de se obter um espaço seguro para o armazenamento dos boletins impressos, demonstraram a necessidade de aperfeiçoar o serviço de emissão e armazenamento dos Boletins de Ocorrência, de forma que o serviço seja interligado entre todas as unidades policiais do Estado.

Tendo em base essa crescente necessidade, a Secretaria de Estado da Segurança Pública e da Defesa Social – SESED do Rio Grande do Norte, por meio da Coordenadoria de Informática e Estatística – COINE, solicitou o desenvolvimento de um sistema de informação, a fim de sanar os problemas enfrentados no serviço de emissão de boletins de ocorrência no Estado.

2 – OBJETIVOS

Este trabalho tem como objetivo geral desenvolver um sistema de informação *web* com o objetivo de sanar os problemas enfrentados pela descentralização do processo de emissão de boletins de ocorrência (B.O), dentre eles a despadronização dos boletins, a falta de informações precisas, a perda de controle na numeração dos B.O. e a falta de local seguro para o armazenamento e fácil acesso aos mesmos.

Para isso, será necessário:

- Descrever os objetivos do sistema de informação “Delegacia Virtual”;
- Descrever o sistema de informação, seus módulos e funcionalidades;
- Explicar as tecnologias utilizadas no desenvolvimento do sistema de informação;
- Detalhar o código-fonte do sistema.

3 – SOLUÇÃO PROPOSTA

A solução proposta para sanar os problemas que os agentes de polícia enfrentavam diariamente com o serviço de emissão de Boletins de Ocorrência foi a criação de um novo Sistema de Informação, chamado de “Delegacia Virtual”. A Delegacia Virtual é um sistema de informação voltado para *web* que tem como principal objetivo melhorar o serviço de emissão de Boletins de Ocorrência, tratados neste trabalho como B.O., que é de responsabilidade da Polícia Civil.

Sua intenção é tornar o processo de emissão do boletim totalmente digital, proporcionando uma maior agilidade na emissão, maior confiabilidade nas informações contidas no B.O. e maior segurança no armazenamento das informações, evitando possíveis duplicidades nas numerações do boletins e possibilitando padronização dos documentos de todas as unidades policiais e a expedição de relatórios estatísticos mais precisos e de forma mais ágil para controle interno.

No sistema, é possível encontrar as seguintes funcionalidades, divididas em módulos:

3.1 – Módulo de Boletins

O Módulo de Boletins é a base de todo o sistema “Delegacia Virtual” e, portanto, é o mais importante. Tendo sido o primeiro módulo a ser desenvolvido, é o mais consolidado. Ele promove a integração entre o sistema da “Delegacia Virtual” e o Sistema de Inquéritos da Polícia Civil, da seguinte maneira: ao se despachar um determinado B.O. para um inquérito policial, automaticamente, os dados contidos no boletim são inseridos no Sistema de Inquérito, gerando um novo inquérito policial. Abaixo, segue a lista das funcionalidades encontradas no módulo:

- **Cadastrar de Boletins:** permite armazenar as informações colhidas durante a formulação do boletim de ocorrência em um banco de dados;

- **Consultar Boletins:** permite que o usuário consulte os boletins de ocorrência armazenados na base de dados utilizando os seguintes critérios de busca: número de protocolo, período do boletim e nome (do comunicante, da mãe, do pai, etc);
- **Movimentar Boletins:** permite o encaminhamento de boletins para outros setores da Unidade Policial em que o usuário está “logado”, ou seja, registrado no sistema (Setor de Investigação, Cartório, ao Delegado ou Arquivamento); encaminhamento do B.O. a outras delegacias; geração de Inquéritos Policiais, Termos Circunstanciados de Ocorrência (TCO) e Boletins de Ocorrência Circunstanciados (BOC), por meio da integração com o Sistema de Inquéritos da Polícia Civil;

3.2 – Módulo de Mandados de Prisão

Um dos módulos mais recentes no sistema, o Módulo de Mandados de Prisão disponibiliza uma consulta fonética utilizando critérios como nome, nome do pai e nome da mãe de pessoas que tenham mandados de prisão/busca e apreensão cadastrados no sistema. Abaixo, segue a lista da funcionalidade disponibilizada no módulo:

- **Realizar Consultas de Mandados de Prisão:** permite que o usuário realize consultas de mandados de prisão/busca e apreensão na base de dados do sistema da Delegacia Especializada em Capturas – DECAP.

3.3 – Módulo de Impressões

Módulo existente no sistema desde a sua primeira versão, o Módulo de Impressões utiliza o *framework* de geração de relatórios, *JasperReports*. Ele permite a geração de todos os relatórios e visualização dos boletins de ocorrência, sempre no formato PDF. Abaixo, segue a lista de suas funcionalidades:

- **Gerar Relatórios Estatísticos em PDF:** ao escolher o período e o tipo de relatório desejado, dentre as opções Ocorrências por *Status*, por Tipo, Cidade/Bairro e Tipos de Ocorrência por Bairro, o sistema exibe uma janela *popup* e o *framework* gera um relatório que permite que o usuário visualize o relatório estatístico, em arquivo PDF;
- **Gerar Visualização em PDF dos Boletins:** ao finalizar o cadastro de um novo B.O., o sistema disponibiliza a opção do usuário visualizar este boletim padronizado para todas as unidades policiais, no formato PDF.

3.4 – Módulo de Homologações

O Módulo de Homologações foi desenvolvido especialmente para a Delegacia do Cidadão – DECIDA, unidade responsável pela homologação de boletins de ocorrência feitos pelo cidadão, utilizando uma interface disponibilizada no próprio *website* da Secretaria de Estado da Segurança Pública e da Defesa Social. Segue, abaixo, a lista das funcionalidades do módulo:

- **Homologar Boletins Feitos pela Internet:** ao acessar o módulo de homologações, o sistema exibe uma listagem de todos os boletins cadastrados pela *internet* que estão aguardando atendimento. O usuário que está “logado”, isto é, registrado no sistema, seleciona um desses boletins e é encaminhado a uma página onde são exibidas todas as informações que foram cadastradas para o boletim. Caso as informações estejam corretas, o usuário pode optar por homologar o boletim. Após a homologação, o cidadão que cadastrou o boletim recebe uma cópia do mesmo em seu *e-mail*;
- **Ignorar Boletins Feitos pela Internet:** de forma similar à funcionalidade anterior, ao analisar o boletim, caso o mesmo não se encaixe nos tipos de ocorrência que podem ser registradas pela *internet* ou não possa ser homologado por quaisquer outras razões, o usuário tem a opção de ignorar o boletim, e informando o motivo para tal. O cidadão que registrou o boletim recebe um *e-mail* informando o motivo pelo qual o seu B.O. foi ignorado;

- **Enviar E-mail ao Cidadão:** caso alguma informação necessária ao B.O. esteja faltando ou o usuário necessite de esclarecimentos em relação a algum dado do boletim, ele pode enviar, por meio desta funcionalidade, um e-mail ao cidadão solicitando esses esclarecimentos.

3.5 – Módulo do CIOSP

Este módulo é responsável pela inédita integração entre o serviço de chamada de emergência (190) e o serviço de registro de boletins de ocorrência. A integração promovida por este módulo se dá nas informações que foram armazenadas ao registrar uma ocorrência pelo telefone com o Centro Integrado de Operações em Segurança Pública – CIOSP, que podem ser vinculadas a um boletim de ocorrência, aumentando, assim a quantidade de informações presentes nele, bem como a precisão das mesmas. Isso porque as ocorrências registradas pelo CIOSP são georeferenciadas, ou seja, contendo precisamente seu local.

Abaixo, segue a lista das funcionalidades presentes no módulo:

- **Exibir Ocorrências Georeferenciadas do Dia em um Mapa Utilizando a API do Google Maps:** por meio da API do *Google Maps*, as ocorrências cadastradas no banco de dados do CIOSP em um determinado dia, que estão georeferenciadas, são exibidas na forma de marcadores no mapa, na sua respectiva georeferência;
- **Listagem de Todas as Ocorrências do Dia:** todas as ocorrências registradas no dia, sejam elas georeferenciadas ou não, são exibidas numa tabela divididas em colunas – Código da Ocorrência, Data/Hora, Nome do Comunicante, Telefone, Endereço – e ordenadas de forma decrescente pela data/hora de cadastro;
- **Consultar Ocorrências:** consultar ocorrências armazenadas na base de dados do CIOSP utilizando um dos seguintes critérios: código da ocorrência, nome do chamador ou período;

- **Vincular Ocorrências:** para uma maior quantidade de informações a respeito de uma determinada ocorrência para a qual venha a ser registrado o B.O., é possível vincular as informações passadas pelo telefone ao CIOSP. Para isso, basta localizar a ocorrência no mapa, na lista ou por meio da consulta, selecioná-la e escolher a opção de “Vincular”. Todos os dados informados ao CIOSP serão automaticamente anexados ao B.O.

3.6 – Módulo do DETRAN

Por meio de um *webservice*, é possível consultar placas de qualquer veículo registrado no Departamento Estadual de Trânsito – DETRAN/RN. Trabalhando com um arquivo *XML* retornado pelo *webservice*, as informações a respeito do veículo são extraídas e disponibilizadas para o usuário, caso a placa do veículo tenha sido localizada no sistema. Abaixo, segue a funcionalidade encontrada no módulo:

- **Consultar Placas de Veículos:** o usuário digita a placa do veículo que deseja consultar e o formulário com as informações a respeito do veículo é automaticamente preenchido.

3.7 – Módulo do ITEP

Por meio de consultas realizadas por qualquer número de carteira de identidade/registro geral (RG) na base de dados do Instituto Técnico-Científico de Polícia - ITEP/RN, são exibidas informações sobre o titular do mesmo. Abaixo, segue a descrição da funcionalidade encontrada no módulo:

- **Consultar RGs:** permite que o usuário digite o número da carteira de identidade/registro geral (RG) do cidadão do Rio Grande do Norte e realize uma consulta. Ao ser localizado o RG, o formulário que contém as informações do cidadão (comunicante, vítima ou investigado) é automaticamente preenchido com os dados encontrados.

4 – REFERENCIAL TEÓRICO

Neste capítulo, serão apresentadas as principais tecnologias utilizadas no desenvolvimento do sistema “Delegacia Virtual”, bem como os seus conceitos e definições.

4.1 – Java

Java é uma linguagem de programação de alto nível, que foi desenvolvida pela empresa *Sun Microsystems* e anunciada formalmente em maio de 1995. A linguagem Java é interpretada, ou seja, seus códigos-fonte são transformados em *byte-code* e então interpretados pela *Java Virtual Machine* – JVM, a máquina virtual Java. A JVM não só é a responsável pela interpretação, como também pela execução desse código-fonte, independente da plataforma em que ele será executado.

Pela existência dessa máquina virtual para vários dispositivos diferentes, com sistemas operacionais distintos, só é necessário escrever todo o sistema uma vez, já que ele poderá ser executado em qualquer ambiente capaz de rodar uma máquina virtual Java.

A Figura 01 ilustra o fluxo para a geração de código-fonte Java em diferentes sistemas operacionais.

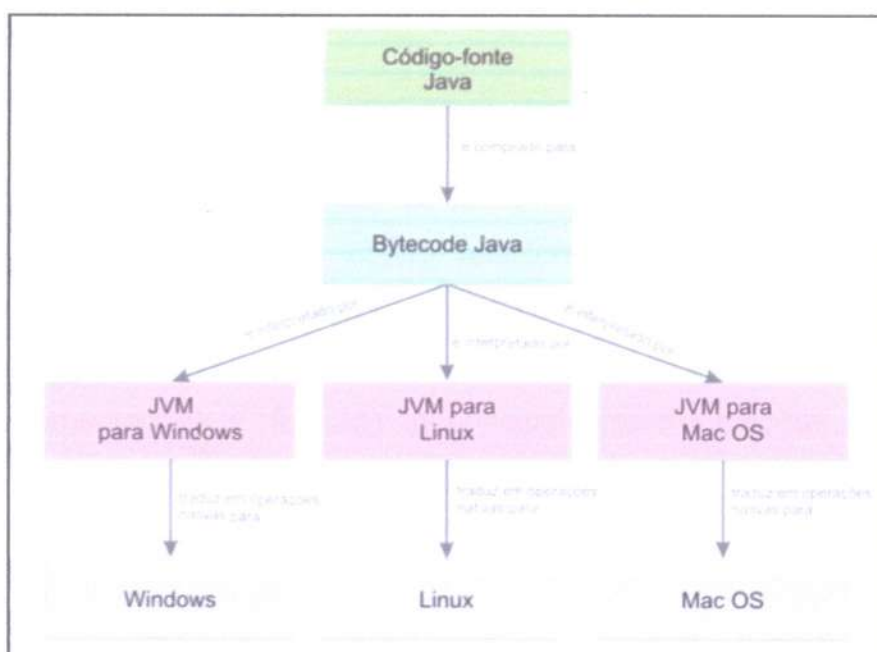


Figura 01 - Fluxo do Código-Fonte. Fonte: ANDRADE, p.18, 2011

A linguagem *Java* foi escolhida para o desenvolvimento do sistema “Delegacia Virtual” por ser uma linguagem consolidada no mercado, com um grande número de profissionais que já a utilizam; por ser portátil; e, finalmente, por ser uma das linguagens normalmente utilizadas no ambiente de desenvolvimento da Coordenadoria de Informática e Estatística da Secretaria de Estado da Segurança Pública e da Defesa Social – SESED.

4.2 – Microsoft SQL Server

O *Microsoft SQL Server* é um Sistema Gerenciador de Banco de Dados Relacional – SGBD que foi desenvolvido pela empresa *Microsoft* em conjunto com a empresa *Sybase Corporation* em 1998, e que, atualmente, é mantido apenas pela *Microsoft*.

Por ser um banco de dados confiável e seguro, o *Microsoft SQL Server 2008* é o SGBD escolhido para armazenar, consultar e manipular todos os dados que vierem a serem armazenados pela “Delegacia Virtual”.

4.3 – Java API Persistence

A *Java API Persistence*, ou JPA, como é mais conhecida, é uma especificação criada com o objetivo de padronizar as ferramentas *Object Relational Mapping* – ORM, isto é, de mapeamento objeto relacional para as aplicações Java, e consequentemente, diminuir a complexidade do desenvolvimento. Por ser apenas uma especificação, a JPA não implementa nenhum código. Para isso, existem diversas implementações (K19, p.38). As principais implementações são: *Hibernate*, *TopLink*, *EclipseLink* e *OpenJPA*.

Para o gerenciamento da camada de persistência da “Delegacia Virtual”, foi selecionada a implementação da especificação JPA *The Eclipse Persistence Service*, também conhecida como *EclipseLink*, desenvolvido e mantido pela *Eclipse Foundation*, organização criada em janeiro de 2004.

Para utilizar essa tecnologia, foram necessárias algumas configurações específicas no projeto:

- Configuração das informações do banco de dados a ser utilizado, entidades que serão gerenciadas pelo *framework* no arquivo *persistence.xml*, como mostrado na Figura 02;

```
<?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="ExemploJPA" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.driver" value="net.sourceforge.jtds.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:jtds:sqlserver://localhost;databaseName=Exemplo" />
      <property name="javax.persistence.jdbc.user" value="exemplo" />
      <property name="javax.persistence.jdbc.password" value="exemplo" />
    </properties>
  </persistence-unit>
</persistence>
```

Figura 02 - Configurando o JPA. Fonte: Resultado da Pesquisa do Autor

- Adição dos arquivos *.jar* necessários para o funcionamento do *framework*, juntamente com o arquivo *.jar* do *driver* para conexão com o banco de dados escolhido, como mostrado na Figura 03;



Figura 03 - JARs para o JPA. Fonte: Resultado da Pesquisa do Autor

- Para que o *framework* gerencie uma tabela, isto é, realize operações com ela, é necessário mapeá-la em uma entidade (classe Java). A Figura 04 mostra um exemplo de uma classe mapeada.

```
@Entity
@Table(name = "Usuarios", schema = "dbo")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_usuario")
    private Long id;

    private String login;

    private String senha;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }
}
```

Figura 04 - Classe Mapeada com JPA. Fonte: Resultado da Pesquisa do Autor

4.4 – Java Server Faces

Segundo Burns (2010, p.3), o *JavaServer Faces* – JSF é um *framework Java* destinado à construção de aplicações *web*. Ele simplifica o desenvolvimento da

interface com o usuário, uma das partes mais difíceis e tediosas do desenvolvimento de aplicações *web*.

O JSF foi definido pelo *Java Community Process* – JCP baseado no padrão de projetos *Model, View, Controller*, ou MVC. Ele utiliza o modelo de interfaces gráficas baseadas em eventos. O padrão MVC, é dividido em três camadas:

- **Model**, que é a camada de Modelo, responsável por representar os objetos presentes nos sistema e fornecer acesso aos dados;
- **View**, a camada de visualização responsável pela interface com o usuário, exibindo componentes para a entrada de dados e ações feitas pelo usuário;
- **Controller**, camada responsável por fazer a interligação entre as camadas de visualização e de modelo, processando as ações que foram enviadas pelo usuário, manipulando os dados do modelo e retornando respostas ao usuário.

A Figura 05 mostra a arquitetura do *JavaServer Faces* utilizando o padrão de projetos MVC.

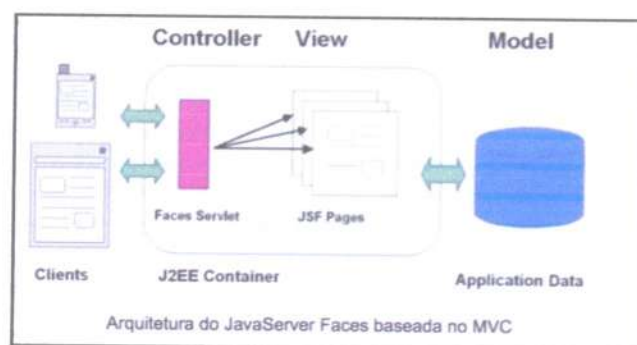


Figura 05 - Arquitetura do *JavaServer Faces*. Fonte: ANDRADE, p.15 (2010)

Para utilizar o JSF, é necessário realizar algumas configurações e procedimentos específicos no projeto:

- Configurar a *Faces Servlet* no *web.xml*, principal arquivo de configurações de um projeto *web Java*, como exibido na Figura 06 abaixo;

```
<!-- Declaração da Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Figura 06 - Configuração da *FacesServlet*. Fonte: Resultado da Pesquisa do Autor

- Definir o padrão de *URLs* no *web.xml*. A Figura 07, exibida abaixo, mostra como definir o padrão;

```
<!-- Definição do padrão das URLs -->
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

Figura 07 - Padrão de *URLs*. Fonte: Resultado da Pesquisa do Autor

- E, por fim, adicionar as bibliotecas, isto é, os arquivos *.jar*, na pasta *WEB-INF/lib*, para que o JSF funcione corretamente. A Figura 08 apresenta os JARs necessários para realizar esse procedimento.

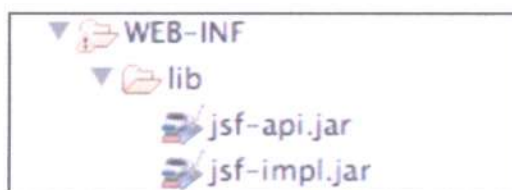


Figura 08 - JARs para o *JavaServer Faces*. Fonte: Resultado da Pesquisa do Autor

Após realizados todos os passos citados acima, é necessário partir para a criação das páginas utilizando *facelets*, no padrão XHTML, necessário para se trabalhar com JSF 2.0.

A Figura 09, exibida abaixo, mostra um simples exemplo de uma página de *login* utilizando o JSF 2.

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">

<body>
  <h2>Exemplo JSF</h2>
  <h:outputText value="Login:" />
  <h:inputText value="#{exemploController.login}" />
  <h:outputText value="Senha:" />
  <h:inputText value="#{exemploController.senha}" />
  <h:commandButton action="#{exemploController.logar()}" value="Logar" />
</body>
</html>

```

Figura 09 - Página com JavaServer Faces. Fonte: Resultado da Pesquisa do Autor

Andrade (2011, p. 56) defende que, ao se requisitar uma página que utiliza componentes JSF, ela passará por um ciclo de vida de processamento bem definido, constituído de seis fases. A Figura 10, abaixo, demonstra como essas fases se organizam.

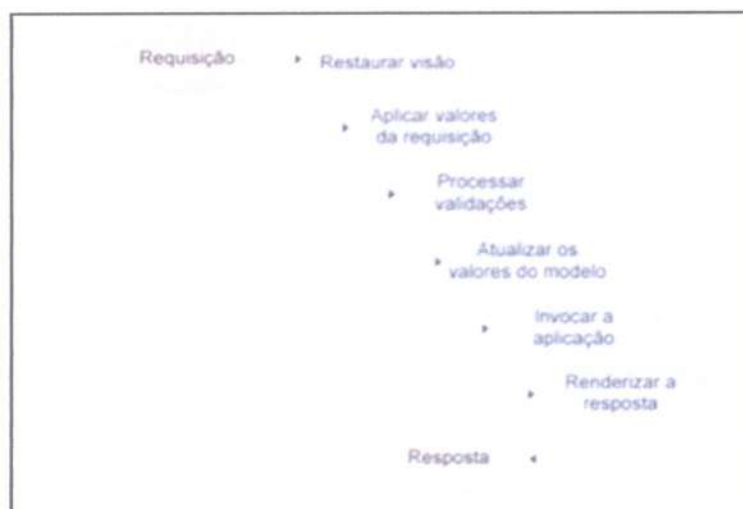


Figura 10 - Ciclo de Vida do JavaServer Faces. Fonte: ANDRADE, p.56 (2011)

A título de explicação, segue o detalhamento do ciclo de vida do JSF:

1. Restaurar Visão:

Por meio da restauração de visão, a árvore de componentes JSF que fazem parte da página solicitada é recuperada, caso já tenha sido exibida, ou constrói-se uma nova árvore, caso seja sua primeira exibição.

Segundo Andrade (2011, p. 56), “caso a página já tenha sido exibida, todos os componentes podem ser recuperados com o seus respectivos estados anteriores, incluindo dados de formulários que foram submetidos ao servidor”.

2. Aplicar Valores de Requisição:

Nesta fase, “cada componente tem a chance de atualizar o seu próprio estado com as informações oriundas da requisição”. (ANDRADE, 2011, p.56)

3. Processar Validações

Ainda segundo Andrade (2011, p. 57), é nesta fase do ciclo de vida do JSF que os validadores e conversores entram em ação. Caso ocorra algum erro de conversão ou validação, o ciclo de vida salta imediatamente para a resposta, pulando todas as outras fases e exibindo a página ao usuário, para que o mesmo possa corrigir e submeter os dados novamente.

4. Atualizar os Valores do Modelo

De acordo com o autor Andrade (2011, p. 57), é nesta fase do ciclo que os valores anexados aos componentes são atualizados nos objetos do modelo de dados.

5. Invocar Aplicação

Novamente de acordo com Andrade (2011, p. 57), nesta etapa os eventos que originaram a submissão do formulário ao servidor são executados.

6. Renderizar a Resposta

“A última fase do ciclo de vida é a renderização da resposta, que gera a saída, com todos os componentes em seus estados atuais, e envia para o cliente”. (ANDRADE, 2011, p. 57)

4.5 – Primefaces

Primefaces é uma biblioteca *open source* de componentes ricos, desenvolvido e mantido pela empresa turca de desenvolvimento de *software* *Prime Teknoloji*

(ÇIVICI, p.12). A biblioteca oferece diversas opções de temas e componentes ricos desenvolvidos para serem usadas em conjunto com o *JavaServer Faces*, com o objetivo de facilitar o desenvolvimento de interfaces com usuário em projetos *web*.

É interessante ressaltar que, para utilizar os componentes do *Primefaces*, também é necessário realizar algumas configurações e procedimentos no projeto e em suas páginas XHTML:

- Adicionar o arquivo *.jar* da biblioteca na pasta WEB-INF/lib. A Figura 11, exibida abaixo, mostra o arquivo *.jar* necessário;

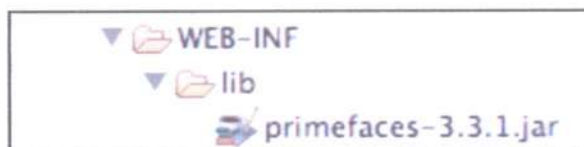


Figura 11 - JARs para o Primefaces. Fonte: Resultado da Pesquisa do Autor

- Adicionar o *namespace* padrão em todas as páginas XHTML presentes no projeto que farão uso de componentes do *Primefaces*. A Figura 12 mostra como deve ser definido o *namespace*;

```
xmlns:p="http://primefaces.org/ui"
```

Figura 12 - Namespace do Primefaces. Fonte: ÇIVICI, p.14

- Caso se deseje utilizar um tema diferente do pré-configurado, deve-se configurar o tema desejado no *web.xml* e adicionar o seu respectivo arquivo *.jar* na pasta WEB-INF/lib do projeto. As Figuras 13 e 14, respectivamente, exemplificam os processos.

```
<!-- Tema do Primefaces -->
<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>redmond</param-value>
</context-param>
```

Figura 13 - Definindo o Tema do Primefaces. Fonte: Resultado da Pesquisa do Autor



Figura 14 - JARs para o Tema do *Primefaces*. Fonte: Resultado da Pesquisa do Autor

4.6 – *JasperReports*

O *JasperReports* é uma biblioteca de geração de relatórios desenvolvida pela empresa *JasperSoft*. Ela tem a habilidade de disponibilizar relatórios com conteúdo rico, seja na tela, diretamente para a impressora ou para arquivos com o formato PDF, HTML, XML, dentre outros.

Segundo Danciu e Chirita (p. 06), a biblioteca é escrita única e exclusivamente em linguagem *Java*, podendo ser utilizada em diversos tipos de aplicações *Java*, incluindo J2EE ou aplicações *web*, gerando, assim, conteúdo dinâmico. Relatórios feitos utilizando essa biblioteca são gerados a partir de *templates* com a extensão *.jrxml*, que posteriormente são compilados, gerando arquivos com a extensão *.jasper*.

O *layout* dos *templates* podem ser criados utilizando *tags* XML próprias do *JasperReports* ou de forma visual, utilizando do recurso “*drag and drop*”, ou “arraste e solte”, presente na ferramenta de *design iReport*.

A Figura 15 ilustra um trecho do *template* utilizado para a geração dos boletins de ocorrência em formato PDF, utilizando as notações XML próprias da biblioteca.

```

<band height="51" splitType="Stretch">
  <image hAlign="Center" vAlign="Middle" onErrorType="Blank">
    <reportElement uuid="faae3cda-2e70-4162-9a54-1eeea2bc3326" x="0" y="0" width="539" height="51"/>
    <imageExpression><![CDATA[{P{topo}}]></imageExpression>
  </image>
</band>
</title>
<detail>
  <band height="11" splitType="Stretch">
    <line>
      <reportElement uuid="d803ee62-b198-46f9-bad6-b1b0ac72f1a1" x="0" y="0" width="539" height="1"/>
    </line>
    <staticText>
      <reportElement uuid="c61deb9a-bd69-440f-b17f-6e72902c22fb" x="0" y="1" width="539" height="9"/>
      <textElement textAlignment="Center" verticalAlignment="Middle">
        <font size="7" isBold="true"/>
      </textElement>
      <text><![CDATA[BOLETIM DE OCORRÊNCIA]]></text>
    </staticText>
    <line>
      <reportElement uuid="88818d52-da70-4600-8350-bef0e5726f2e" x="0" y="10" width="539" height="1"/>
    </line>
  </band>

```

Figura 15 - *Template de Relatório em XML. Fonte: Resultado da Pesquisa do Autor*

A título de explicação, é interessante frisar que o *iReport Designer* é uma ferramenta grátis e *open source*, desenvolvida em Java pela JasperSoft, que permite a montagem de *templates* de relatórios utilizados pelo JasperReports e JasperReports Server. O *iReport* suporta diversos componentes distintos, como tabelas, listas, gráficos e mapas.

A Figura 16, apresentada abaixo, ilustra o *template* utilizado para a emissão dos boletins de ocorrência em formato PDF na interface do *iReport*.

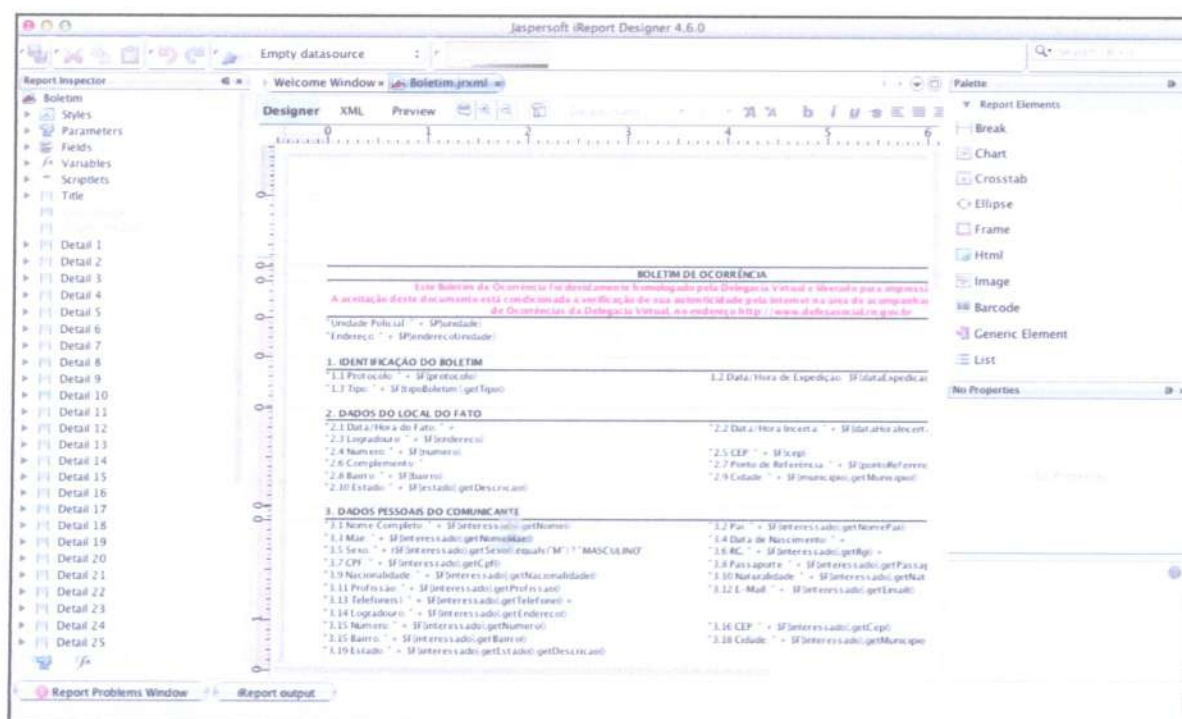


Figura 16 - Template de Relatório no iReport. Fonte: Resultado da Pesquisa do Autor

Já a Figura 17 ilustra um método simplificado, responsável pela geração da visualização de um boletim de ocorrência, no formato PDF, para ser exibido no navegador do usuário.

```
private void imprimir() {
    //HashMap que guarda os parâmetros passados ao relatório.
    Map<String, Object> parametros = new HashMap<String, Object>();

    //Classe responsável pela formatação de datas.
    SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

    // Collection que será transformada em DataSource
    Collection<Boletim> boletins = new ArrayList<Boletim>();
    boletins.add(boletim);

    //... Preenchendo o HashMap com os parâmetros que são passados para o relatório ...

    //Preparando a resposta para exibir conteúdo no formato PDF.
    HttpServletResponse response = (HttpServletResponse) FacesContext
        .getCurrentInstance().getExternalContext().getResponse();
    response.setContentType("application/pdf");
    response.setHeader("Content-disposition", "filename=Boletim.pdf");
    try {
        //Gerando o DataSource a partir da Collection de boletins.
        JRDataSource dataset = new JRBeanCollectionDataSource(boletins);

        //Montando o relatório a partir do modelo de impressão (arquivo .jrxml), parâmetros e o DataSource.
        JasperPrint print = JasperFillManager.fillReport(modeloImpressao,
            parametros, dataset);

        //Exportando o relatório para um PdfStream, para ser exibido no navegador.
        JasperExportManager.exportReportToPdfStream(print,
            response.getOutputStream());

        response.getOutputStream().flush();
        response.getOutputStream().close();

    } catch (Exception e) {
        e.printStackTrace();
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
            "Não foi possível gerar o relatório", null);
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
    FacesContext.getCurrentInstance().renderResponse();
    FacesContext.getCurrentInstance().responseComplete();
}
```

Figura 17 - Código-Fonte para Geração de um Relatório JasperReports. Fonte:
Resultado da Pesquisa do Autor

4.7 – Apache Tomcat

O *Apache Tomcat* é uma implementação *open source* das tecnologias *Java Servlet* e *JavaServer Pages*, desenvolvida pela *Apache Software Foundation*. Atualmente, ela está em sua versão 7.

Apesar de ser um servidor *web* e de rodar aplicações da plataforma JEE, ele não possui suporte a EJBs. Além disso, o *Tomcat* não é o único *container web* para *Java*. Existem, atualmente, outros *containers*, como o *Glassfish*, *Jetty*, *JBoss Application Server* e *WebLogic Application Server*.

Apesar de não ser o único servidor de aplicação disponível para uso, o *Tomcat* foi o escolhido para ser utilizado neste projeto devido ao fato de não ser necessário um servidor de grande porte como o *JBoss AS*, já que o sistema não trabalha com EJBs; e devido ao fato do *Tomcat* ser um servidor bastante utilizado atualmente, o que permite o fácil acesso ao seu suporte técnico.

A Figura 18 ilustra o processo de instalação do *Apache Tomcat 7* como serviço no *Windows Server*.

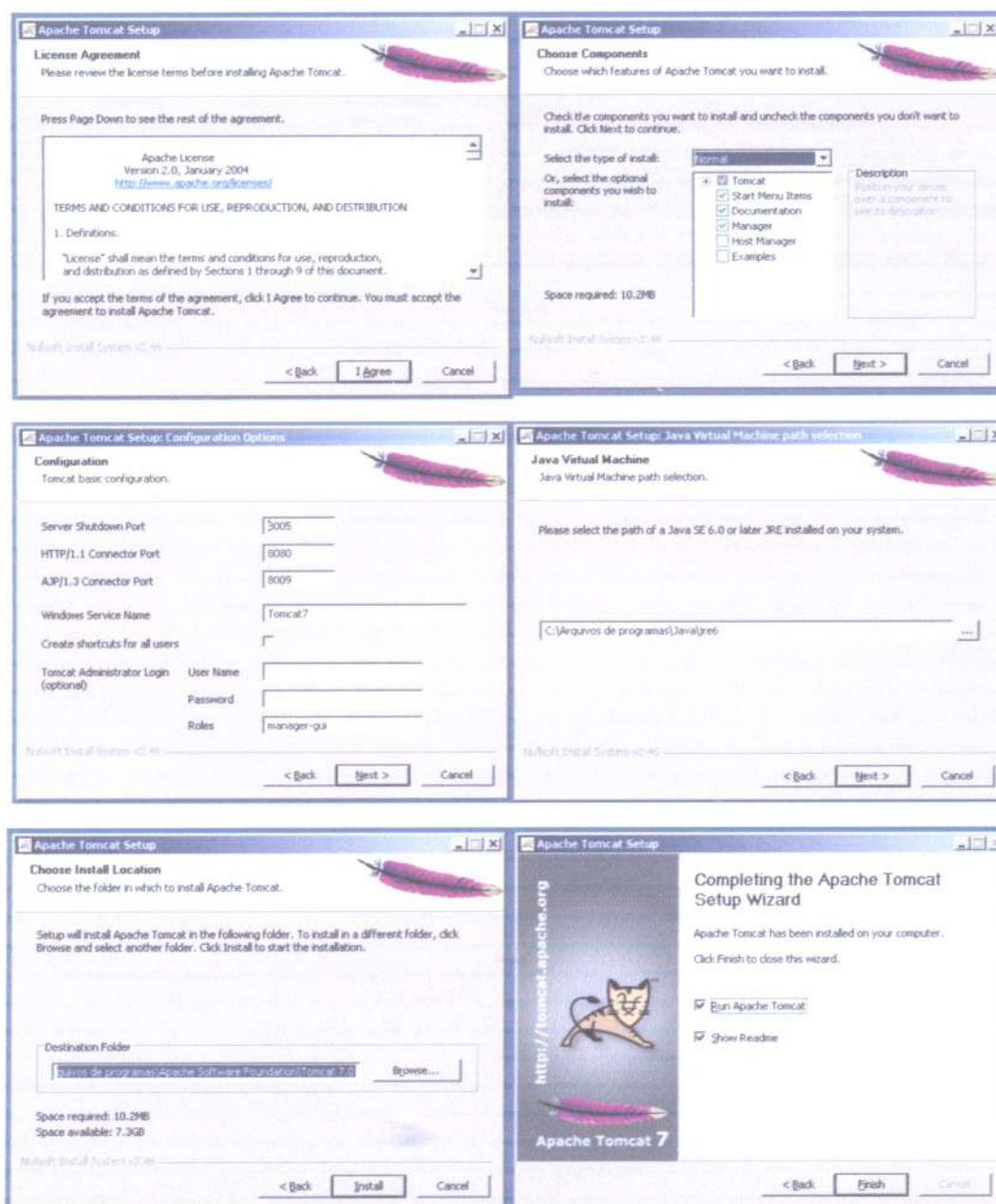


Figura 18 - Instalação do *Apache Tomcat* como Serviço. Fonte: Resultado da Pesquisa do Autor

A Figura 19, por sua vez, ilustra a interface *web* disponibilizada pelo Apache Tomcat para gerenciar as aplicações que estão rodando nele.

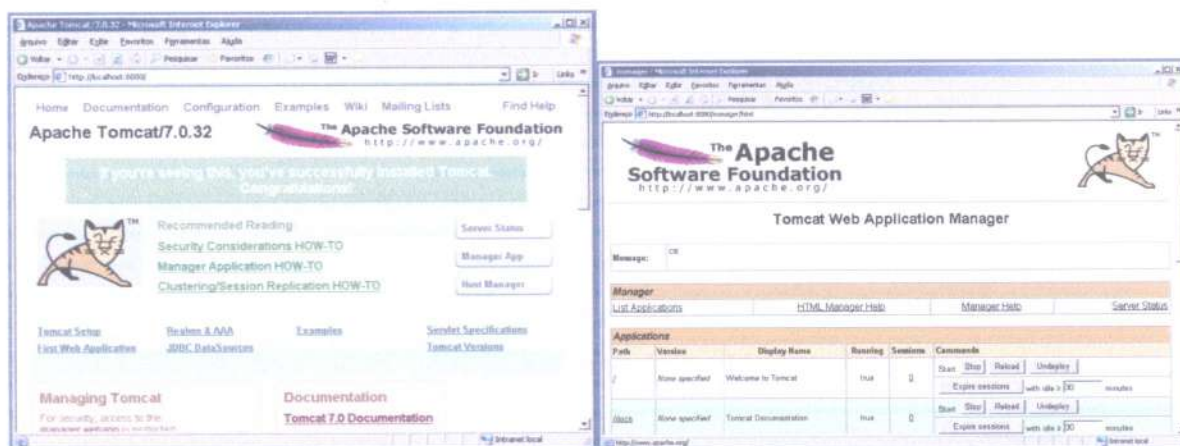


Figura 19 - Interface *Web* para Gerenciar Aplicações. Fonte: Resultado da Pesquisa do Autor

5 – ESTUDO DE CASO

Neste capítulo, será feito o detalhamento do cronograma de desenvolvimento do projeto, como também do código-fonte do sistema de informação “Delegacia Virtual”. Para iniciar o estudo, é necessário destacar os pacotes em que o código-fonte foi dividido, a fim de possibilitar uma melhor organização:

5.1 - Cronograma de Desenvolvimento

Na Tabela 01, exibida abaixo, é apresentado o cronograma de desenvolvimento do projeto, com as atividades realizadas desde a sua concepção.

Fases do Desenvolvimento (Atividades)	Meses (2011)		Meses (2012)										
	11	12	1	2	3	4	5	6	7	8	9	10	11
1 - Requisitos													
2 - Desenvolvimento da Aplicação													
3 - Publicação da Aplicação													
4 - Ajustes na aplicação													
5 - Implementação de novos requisitos													

Tabela 01 – Cronograma de Desenvolvimento do Sistema. Fonte: Resultado da Pesquisa do Autor

Abaixo segue o detalhamento do cronograma:

5.1.1. Requisitos: O projeto “Delegacia Virtual” iniciou-se, de fato, no dia 11 de novembro de 2011, quando foram passados os primeiros requisitos e o diagrama-entidade-relacionamento (DER) do banco de dados que já estava preparado;

5.1.2. Desenvolvimento da Aplicação: A partir do DER do banco de dados e dos primeiros requisitos que foram passados, iniciou-se o desenvolvimento do projeto

também no dia 11 de novembro de 2011, estendendo-se até o início do mês de fevereiro do ano seguinte;

5.1.3. Publicação da Aplicação: Após finalizado o desenvolvimento da aplicação, sua primeira versão estável foi publicada, contendo apenas os seguintes módulos implementados:

- Módulo de Boletins;
- Módulo de Impressões (sem os relatórios estatísticos);
- Módulo do ITEP.

5.1.4. Ajustes na Aplicação: A publicação da aplicação gerou um *feedback* positivo dos usuários:

- Sugestão de Ajustes;
- Problemas Relatados;
- Novos Requisitos.

A partir do *feedback* gerado pelos usuários, iniciaram-se os ajustes necessários na aplicação, que se estenderam até o dia 04 de abril de 2012.

5.1.5. Implementação de Novos Requisitos: Na medida em que novos requisitos e/ou ajustes ficavam prontos, eles foram implementados, novas versões foram liberadas e a aplicação foi crescendo. Atualmente, encontra-se na versão 1.7.4, contendo todos os módulos citados na seção anterior totalmente implementados e em funcionamento.

A figura 20 ilustra um diagrama com os casos de uso que estão presentes no sistema.

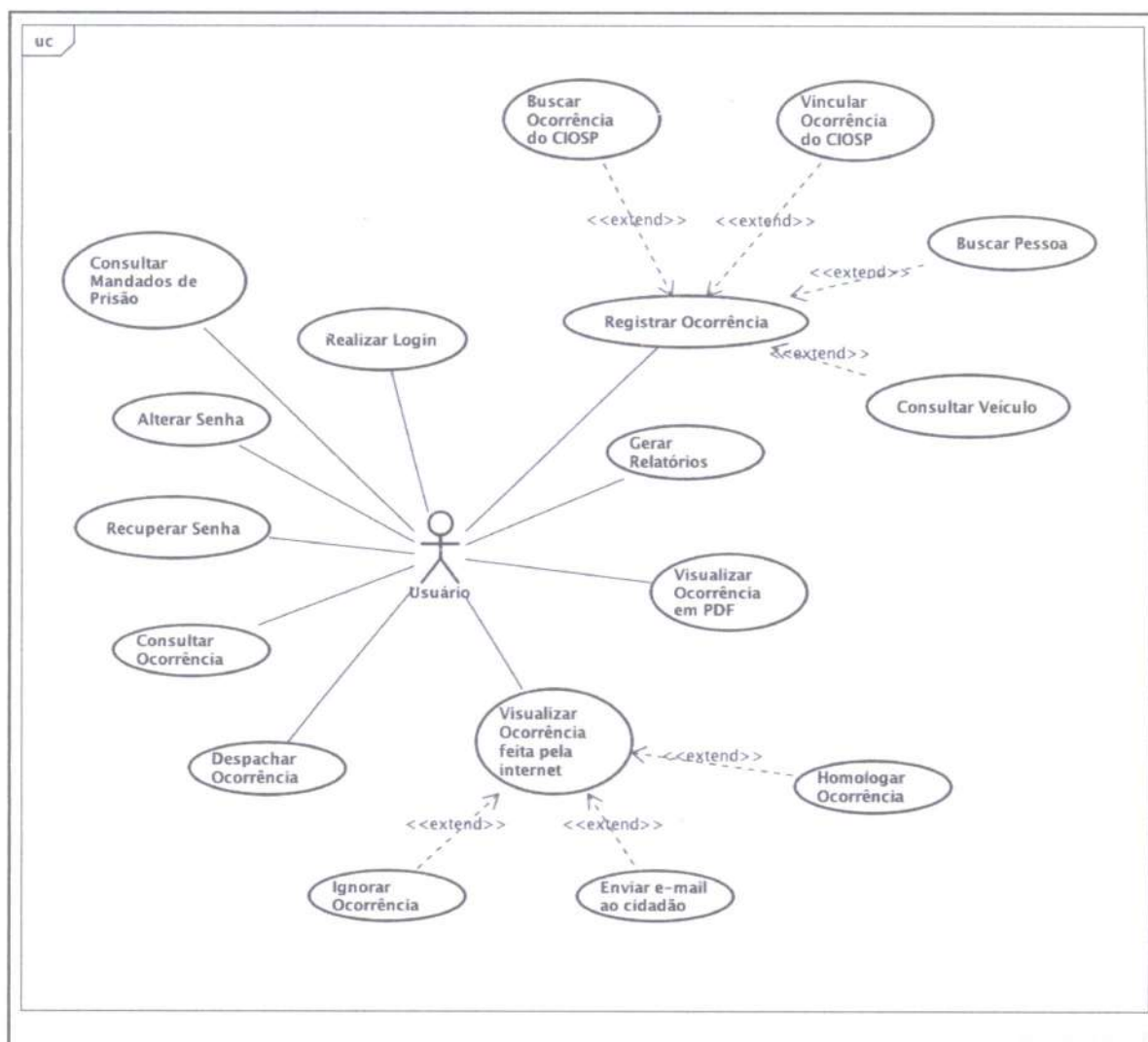


Figura 20 - Diagrama de Casos de Uso. Fonte: Resultado da Pesquisa do Autor

5.2 - Detalhamento do Código-Fonte

- br.dpvirtual.apis;
- br.dpvirtual.controller;
- br.dpvirtual.converter;
- br.dpvirtual.dao;
- br.dpvirtual.filter;
- br.dpvirtual.interfaces;
- br.dpvirtual.listing;
- br.dpvirtual.model;
- br.dpvirtual.utils;

- br.dpvirtual.views;
- br.dpvirtual.webservices.

A Figura 21 ilustra a estrutura do projeto, com ênfase na divisão dos pacotes do código-fonte.

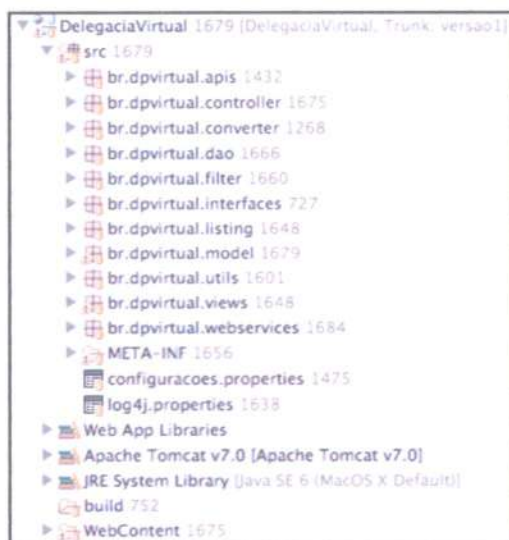


Figura 21 - Estrutura do Projeto no Eclipse. Fonte: Resultado da Pesquisa do Autor

A título de descrição do processo de organização do código-fonte, foi feito, abaixo, o detalhamento de todos os pacotes do projeto, descrevendo para que servem, como funcionam e que classes fazem parte deles.

5.2.1 – Classes de Persistência (br.dpvirtual.dao)

Este pacote é o responsável por toda a camada de persistência do sistema, utilizando o padrão de projeto *Data Access Object* – DAO, todas as classes herdam da superclasse abstrata *GenericDao*, responsável pela implementação das quatro operações básicas na camada de persistência, conhecidas como *Create, Read, Update and Delete* – CRUD, ou “Criar, Ler, Atualizar e Excluir”.

A Figura 22 ilustra a estrutura da superclasse abstrata *GenericDao*, responsável pelas operações do CRUD.

```

public abstract class GenericDao<T> extends PersistDB {
    private EntityManager entityManager;
    private Class<T> objClass;

    public GenericDao(EntityManager entityManager, Class<T> objClass) {
        this.entityManager = entityManager;
        this.objClass = objClass;
    }

    public void create(PersistDB persist) throws SQLException {
        try {
            entityManager.getTransaction().begin();
            entityManager.persist(persist);
        } catch (Exception e) {
            entityManager.getTransaction().rollback();
            throw new SQLException();
        } finally {
            entityManager.getTransaction().commit();
            entityManager.close();
        }
    }

    public void update(PersistDB persist) {
        try {
            entityManager.getTransaction().begin();
            entityManager.merge(persist);
        } catch (Exception e) {
            entityManager.getTransaction().rollback();
        } finally {
            entityManager.getTransaction().commit();
            entityManager.close();
        }
    }

    public T findById(int id) {
        return entityManager.find(objClass, id);
    }

    public Number count() {
        return (Number) entityManager.createQuery(
            "SELECT COUNT(o) FROM " + objClass.getSimpleName() + " o")
            .getSingleResult();
    }

    @SuppressWarnings("unchecked")
    public List<T> getAll() {
        return entityManager.createQuery(
            "SELECT o FROM " + objClass.getSimpleName() + " o").getResultList();
    }
}

```

Figura 22 - Classe *GenericDao*. Fonte: Resultado da Pesquisa do Autor

5.2.2 – Classes de Modelo (br.dpvirtual.model & br.dpvirtual.views)

Os pacotes *br.dpvirtual.model* e *br.dpvirtual.views* são os pacotes responsáveis pela camada de modelo da aplicação. No pacote *br.dpvirtual.model*, é possível encontrar as Entidades, ou classes *Java* com anotações que mapeiam uma tabela da base de dados, que representam as tabelas presentes na base de dados. É interessante ressaltar que todas essas entidades implementam uma mesma interface *Java*, a interface *PersistDB*, que será detalhada mais adiante, neste mesmo trabalho.

O pacote *br.dpvirtual.views* representa as *views* da base de dados que são utilizadas no sistema. As classes deste pacote também implementam a interface *PersistDB*, e sua principal utilização é nas listagens e consultas realizadas no sistema.

A Figura 23 ilustra a entidade “Município” do pacote *br.dpvirtual.model*.

```

@Entity
@Table(name = "dbo.Municipios")
public class Municipio implements Serializable, PersistDB {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CP_MUNICIPIOS")
    private int id;

    @Column(name = "ST_MUNICIPIO")
    private String municipio;

    @Column(name = "ST_UF")
    private String uf;

    @ManyToOne
    @JoinColumn(name = "FK_UF")
    private Estado estado;

    public Municipio() {
        estado = new Estado();
    }

    //Gets & Sets.

```

Figura 23 - Entidade do Pacote br.dpvirtual.model. Fonte: Resultado da Pesquisa do Autor

A Figura 24, por sua vez, ilustra o diagrama de classes presente no pacote br.dpvirtual.models.

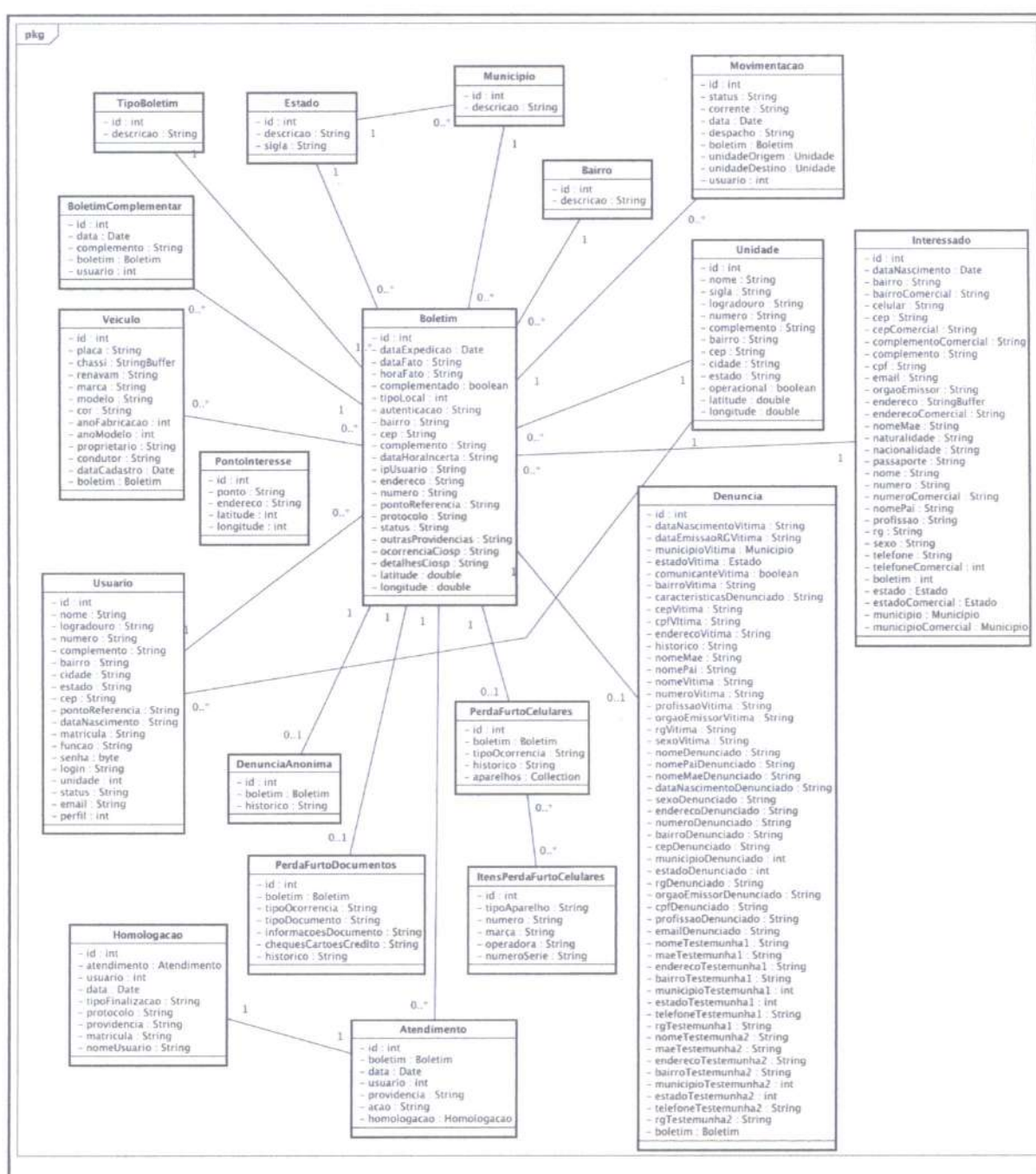


Figura 24 - Diagrama de Classes. Fonte: Resultado da Pesquisa do Autor

5.2.3 – Classes de Controle (br.dpvirtual.controller)

O pacote `br.dpvirtual.controller` é o pacote responsável pelas regras de negócio do sistema. Representa a camada de Controle, formada por *Managed Beans*, classes responsáveis pela comunicação da camada de *view* – páginas com componentes JSF – e a camada de modelo.

A Figura 25, exibida abaixo, exibe o *Managed Bean* responsável pelas buscas de boletins.

```
@ViewScoped
@ManagedBean(name = "buscaController")
public class BuscaController implements Serializable {

    private static final long serialVersionUID = 1L;

    private String protocolo;
    private String nome;
    private Date dataInicial;
    private Date dataFinal;
    private boolean buscaTodasUnidades;
    private Collection<VwListaBoletinsDelegaciasPessoa> listaBusca;

    public void buscarBoletim() {}

    /*
     * Gets and Sets
     */
}
```

Figura 25 - *Managed Bean* Responsável pelas Buscas. Fonte: Resultado da Pesquisa do Autor

5.2.4 - Classes Utilitárias (br.dpvirtual.utils)

Este pacote contém classes utilitárias, auxiliando algumas das funcionalidades mais importantes do sistema. Abaixo, segue a lista de classes pertencentes ao pacote, com suas devidas funções.

- **Criptografia**

Responsável pela geração do *hash* de autenticação de cada boletim de ocorrência que for cadastrado no sistema, utilizando o algoritmo MD5;

- **Formatacao**

Responsável por formatar *strings* para os padrões existentes no banco de dados;

- **GeolocalizationUtil**

Responsável por converter a latitude e a longitude das ocorrências do CIOSP do padrão UTM para o padrão utilizado pelo *Google Maps*, o padrão *Lat/Lng*;

- **PropertiesUtil**

Responsável pela leitura do arquivo de propriedades que contém as configurações do sistema, como por exemplo, dos bancos de dados dos ambientes de testes e de produção;

- **RandomAlphaNumeric**

Responsável pela geração de *strings* aleatórias utilizando caracteres alfanuméricos. Utilizado para recuperar a senha de usuários e enviar uma nova senha temporária ao mesmo;

- **Soundex**

Ao ser realizada uma busca de um mandado de prisão, a classe *Soundex* recebe a *string* e a extrai para, depois, realizar a consulta no banco de dados;

- **Validacao**

Responsável pela validação de dados como CPF, CNPJ, datas e horas.

5.2.5 – Filtros (br.dpvirtual.filter)

Este pacote contém classes *Java* que implementam a interface *Filter* do pacote *javax.servlet*. Por meio da implementação desta interface, essas classes monitoram um determinado padrão de *URLs* configurado no *web.xml* da aplicação, bloqueando, assim, qualquer acesso indevido que se encaixem no padrão de *URLs* configurado.

A Figura 26 mostra a declaração de um filtro no *web.xml* e a configuração do padrão de *URLs* que ele pode monitorar.

```

<!-- Filtro que controla o login no módulo desktop -->
<filter>
  <display-name>ControleSessaoFilter</display-name>
  <filter-name>ControleSessaoFilter</filter-name>
  <filter-class>br.dpvirtual.filter.ControleSessaoFilter</filter-class>
  <init-param>
    <param-name>loginURI</param-name>
    <param-value>/desktop/pages/login.jsf</param-value>
  </init-param>
</filter>

<!-- Mapeamento do filtro do módulo desktop -->
<filter-mapping>
  <filter-name>ControleSessaoFilter</filter-name>
  <url-pattern>/desktop/pages/*</url-pattern>
</filter-mapping>

```

Figura 26 - Declaração de Filtro no web.xml. Fonte: Resultado da Pesquisa do Autor

5.2.6 – Conversores (br.dpvirtual.converter)

As classes deste pacote são responsáveis por disponibilizar o serviço de conversores. Para utilizar esse serviço, é necessário apenas implementar a interface *Converter* do pacote *javax.faces.convert*. É interessante frisar que na versão 1.2 do JSF, era necessário declarar o conversor no arquivo de configurações *faces-config.xml*. Porém, a partir da versão 2, esse processo não se faz mais necessário, bastando utilizar apenas a anotação *@FacesConverter*.

Por meio desse serviço, é possível converter dados preenchidos ou selecionados pelo usuário para o formato necessário para sua manipulação posterior.

A Figura 27 ilustra a implementação de um conversor presente no sistema.

```

@FacesConverter(value = "estadoConverter", forClass = Estado.class)
public class EstadoConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext facesContext, UIComponent component,
        String value) {
        if (value != null && value.matches("[0-9]+")) {
            EstadoDao dao = new EstadoDao();
            return dao.findById(Integer.valueOf(value));
        }
        return null;
    }

    @Override
    public String getAsString(FacesContext arg0, UIComponent arg1, Object value) {
        if (value != null) {
            Estado e = new Estado();
            e = (Estado) value;
            return String.valueOf(e.getId());
        }
        return null;
    }
}

```

Figura 27 - Conversor Implementado no Sistema. Fonte: Resultado da Pesquisa do Autor

5.2.7 – Interfaces (br.dpvirtual.interfaces)

Este pacote contém apenas uma classe, a interface *PersistDB*. Por meio da implementação desta interface, é possível persistir objetos das classes modelo que a implementam.

A Figura 28 demonstra a implementação da interface *PersistDB*.

```

public interface PersistDB {

    public int getId();

    public void setId(int id);

}

```

Figura 28 - Interface *PersistDB*. Fonte: Resultado da Pesquisa do Autor

5.2.8 – APIs (br.dpvirtual.apis)

A classe contida neste pacote faz o uso de duas APIs do Google. As consultas realizadas são retornadas no padrão JSON ou XML e o formato escolhido para a

manipulação dos dados recebidos foi o JSON, por sua estrutura ser mais simples do que os XMLs que são retornados pelas APIs.

Abaixo, segue uma lista das APIs utilizadas e seus locais de utilização:

- **Google Places**

Esta API é utilizada para melhorar a precisão dos endereços onde ocorreram os fatos registrados nos boletins, por meio de uma pesquisa por um ponto de interesse (ex.: *Midway Mall*). A API retornará o endereço completo do ponto de interesse, juntamente com a sua latitude e longitude, que serão armazenados no boletim de ocorrência;

- **Google Reverse Geocoding**

A utilização desta API é de forma similar à anterior, porém ao invés de pesquisar por pontos de interesse, a pesquisa é feita diretamente pelos logradouros, também retornando a latitude e a longitude do local.

5.2.9 – Webservices (br.dpvirtual.webservices)

Este pacote contém apenas uma classe, por meio da qual é possível realizar consultas de placas de veículos utilizando um determinado *webservice* disponibilizado. Por meio da manipulação de um arquivo XML, os dados são extraídos e retornados para os usuários.

Esse serviço mostra-se de grande utilidade para as delegacias, principalmente para aquelas especializadas em ocorrências com veículos, permitindo que os profissionais verifiquem se um determinado veículo possui multas registradas, impedimentos ou até mesmo se possui alguma queixa de roubo relacionada a ele.

5.2.10 – Lazy Listing (br.dpvirtual.listing)

Contendo, também, apenas uma classe, ela é responsável pela principal listagem do sistema, a listagem de todos os boletins que estão pendentes na Delegacia na qual o usuário está “logado”, isto é, registrado no sistema.

A listagem implementada pela classe *BoletimLazyListing* não é uma listagem comum, e sim uma listagem sob demanda, também conhecida como *LazyListing*. Diferentemente das listagens comuns, na *Lazylisting* apenas as páginas que são selecionadas são carregadas, diminuindo, assim, a carga do lado do servidor.

Para que se utilize tal modo de listagem, é necessário que a classe que irá implementá-la estenda a classe *LazyDataModel* do pacote *org.primefaces.model*, em caso de utilização do *Primefaces*.

A Figura 29 ilustra a implementação do *LazyListing* pela classe *BoletimLazyListing*.

```
public class BoletimLazyList extends LazyDataModel<VwListaBoletinsDelegacias> {

    private static final long serialVersionUID = 1L;

    @Override
    public List<VwListaBoletinsDelegacias> load(int first, int pageSize,
        String sortField, SortOrder sortOrder, Map<String, String> filters) {

        VwListaBoletinsDelegaciasDao dao = new VwListaBoletinsDelegaciasDao();
        List<VwListaBoletinsDelegacias> boletins = null;
        Usuario usuario = LoginController.getUsuarioLogado();

        if (usuario != null) {
            boletins = dao.getAll(usuario.getUnidade().getId(), first,
                pageSize, filters);

            if (getRowCount() <= 0)
                setRowCount(dao.count(usuario.getUnidade().getId()));

            setPageSize(pageSize);
            return boletins;
        }

        return boletins;
    }
}
```

Figura 29 - Implementação do *LazyListing*. Fonte: Resultado da Pesquisa do Autor

6 – CONCLUSÃO

A implantação do sistema de informação “Delegacia Virtual” foi efetiva e eficaz, atendendo, desde a publicação de sua primeira versão, a todos os requisitos solicitados por seus usuários. O sistema atendeu de forma plena a necessidade da Secretaria de Estado da Segurança Pública e da Defesa Social – SESED do Rio Grande do Norte, proporcionando aos profissionais da vinculados à segurança pública do Estado uma maior agilidade, confiabilidade, precisão e segurança no processo de registro e emissão de boletins de ocorrência, bem como de cadastro e armazenamento de suas informações.

Além de todos os benefícios supramencionados, foi possível, por meio da utilização diária do sistema, manter de forma prática e simples um controle preciso da quantidade de ocorrências registradas pelas Delegacias de Polícia presentes nos mais diversos municípios do Estado.

A integração dos serviços de chamadas de emergência (190), boletins de ocorrência e inquéritos policiais, que foi implementada no sistema, ajudou a Polícia Civil a obter informações relacionadas às suas investigações de forma mais rápida e segura, visto que passou a ser possível vincular informações obtidas em contato telefônico pelo chamado de emergência a um B.O., bem como transformá-lo em um Inquérito Policial, disponibilizando todas essas informações de forma automatizada.

7 – REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, Thiago Faria de. **Desenvolvimento Web com JavaServer Faces**. 2 ed. Setembro de 2010. Disponível em: <<http://www.algaworks.com/downloads/apostilas/algaworks-dwjsf-desenvolvimento-web-com-javascript-faces-2a-edicao.pdf>>. Acesso em 08 nov. 2012.

ANDRADE, Thiago Faria de. **Fundamentos Java e Orientação a Objetos**. 3 ed. Setembro de 2011. Disponível em: <<http://www.algaworks.com/downloads/apostilas/algaworks-fjoo-fundamentos-java-e-orientacao-a-objetos-3a-edicao.pdf>>. Acesso em 08 nov. 2012.

BURNS, Ed; SCHALK, Chris. **The Complete Reference Guide: JavaServer Faces 2.0**. EUA, McGraw-Hill, 2010.

ÇIVICI, Çağatay. **Primefaces User's Guide 3.2**. Disponível em: <http://primefaces.googlecode.com/files/primefaces_users_guide_3_2.pdf>. Acesso em 18 nov. 2012.

DANCIU, Teodor; CHIRITA, Lucian. **The JasperReports Ultimate Guide**. Disponível em: <<https://www.jaspersoft.com/store/jasperreports-ultimate-guide-documentation>>. Acesso em 19 nov. 2012.

TREINAMENTOS, K19. **Desenvolvimento Web com JSF2 e JPA2**. 2011. Disponível em: <<http://www.k19.com.br/downloads/apostilas/java/k19-k12-desenvolvimento-web-com-jsf2-e-jpa2>>. Acesso em 18 nov. 2012.