DESENVOLVIMENTO DE UMA CAMADA DE ACESSO AOS DADOS AMBIENTAIS E IMAGENS DE SATÉLITE WEBSERVICE AMBIENTAL

Letícia Capucho Luiz (FATEC Cruzeiro, Bolsista PIBITI/CNPq)
E-mail: leticia.luiz@cptec.inpe.br

Daniel Alejandro Vila (DSA/CPTEC/INPE, Orientador)
E-mail: daniel.vila@cptec.inpe.br

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO TECNOLÓGICA (PIBITI/CNPq/INPE)

COLABORADOR

Mário Lemes de Figueiredo Neto (DSA/CPTEC/INPE)

Julho de 2017

RELATÓRIO FINAL DE INICIAÇÃO TECNOLÓGICA DO PROGRAMA: PIBITI/ INPE - CNPq

PROJETO

DESENVOLVIMENTO DE UMA CAMADA DE ACESSO AOS DADOS AMBIENTAIS E IMAGENS DE SATÉLITE WEBSERVICE AMBIENTAL

PROCESSO: 113926/2016-2

Relatório elaborado por Letícia Capucho Luiz relativo ao período de março de 2016 a julho de 2017

Letícia Capucho Luiz – Bolsista PIBITI/CNPq
E-mail: leticia.luiz@cptec.inpe.br

Daniel Alejandro Vila – Orientador
DSA/CPTEC/INPE
E-mail: daniel.vila@cptec.inpe.br

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, porque sem Ele nada seria possível.

Ao meu orientador Daniel Alejandro Vila pela paciência e atenção. Ao colaborador Mário Lemes de Figueiredo Neto, pelos conselhos. Por fim, aos amigos e família, por terem me motivado e apoiado as minhas decisões.

LISTA DE FIGURAS

RESUMO

O presente trabalho consiste em desenvolver uma camada padronizada de acesso aos dados ambientais e imagens de satélites, utilizando-se da tecnologia de web service, que tem como objetivo a integração de sistemas entre diferentes aplicações. Com esta tecnologia é possível que novas aplicações possam interagir com outras e que os sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Ressaltando, com a utilização de um web service ambiental permite à DSA(Divisão de Satélites e Sistemas Ambientais) que as suas aplicações enviem e recebam dados em formatos padronizados como *XML* ou *JSON*. Sendo utilizado também a *Representação de transferência de Status – REST*, ao qual trabalha com a disponibilização de dados no formato *JSON*, que significa, Notação de Objetos *JavaScript*, permitindo a troca de informações dos dados em tempo real.

ABSTRACT

The present work consists of a standardized solution of access to environmental data and satellite images, using web service technology, which aims to integrate systems between different applications. With this technology it is possible that new applications interact with others and which systems developed in different and compatible platforms. The use of an environmental web service allows the DSA (Division of Satellites and Environmental Systems) to send and receive data in standardized formats such as *XML* or *JSON* as their applications. It is also used a *Status Transfer Representation - REST,* to work with a raw data availability *JSON*, which means, *JavaScript* Object Notation, allowing a real-time exchange of information of the data.

Sumário

1. INTRODUÇÃO

O presente relatório tem por objetivo demonstrar o funcionamento e os resultados obtidos através da utilização da tecnologia de *Web Service,* onde o mesmo é trabalhado com conceitos de computação distribuída e camadas de serviços, a fim de que o processamento das requisições possam ser divididos entre os equipamentos do ambiente computacional.

Tendo em vista da utilização destes conceitos de computação distribuída, foi pensado em utilizar-se de um *Web Service Ambiental*, cujo, o desenvolvimento de uma camada padronizada de acesso aos dados ambientais e imagens de satélite, de diferentes fontes e tipos de dados, e disponibilizá-los em diferentes formatos para a interação com outros softwares.

Entretanto, para uma melhor performance o projeto foi direcionado à *Web Services REST* (*Representação de transferência de Status*) que trabalham com a disponibilização de dados no formato *JSON*, que significa, *Notação de Objetos JavaScript,* permitindo o retorno dos dados em tempo real.

2. OBJETIVOS

2.1 Objetivo Geral

Desenvolver uma camada de acesso padronizada, aos dados ambientais e imagens de satélites, utilizando-se da tecnologia de *Web Service*, que permite a integração de sistemas distribuídos.

2.2 Objetivos Específicos

A padronização dos formatos de retorno aos dados oferecidos pelo CPTEC, através da Divisão de Satélites e Sistemas Ambientais, permitindo que os usuários e comunidade científica o fácil acesso a esses dados utilizando um padrão homogêneo.O principal formato adotado para integração desses sistemas e acesso a esses dados é o *JSON (Notação de Objeto JavaScript).*

3. APLICAÇÕES UTILIZADAS

De acordo com as necessidades apresentadas para a criação do *Web Service* foi proposta a utilização da Linguagem de Programação JAVA em conjunto com a IDE Eclipse para o desenvolvimento da aplicação, e da base de dados do *PostgreSQL,* utilizando-se da interface gráfica pgAdmin III ao qual se encontra os dados de banco.

Com a finalidade de que a aplicação enviem e recebam dados em formatos padronizados como *XML* ou *JSON,* foi utilizado a *Representação de transferência de Status – REST*, ao qual trabalha com a disponibilização de dados no formato *JSON*, que significa, Notação de Objetos *JavaScript*, permitindo a troca de informações dos dados em tempo real.

3.1 JAVA

Uma linguagem de programação interpretada e orientada a objetos, que não contém redundâncias, de fácil entendimento, implementação e de utilizar. Robusta e fortemente tipada, sua declaração é do tipo obrigatória, a fim de obter uma declaração de qual será o tipo de objeto que a variável irá se referir. Além disto, possui portabilidade e é encontrada em aplicações as quais funcionam do mesmo jeito em qualquer ambiente, sendo completamente especificada.

3.2 PostgreSQL

É um Sistema Gerenciador de Banco de Dados relacional de código aberto, ou seja, o PostgreSQL é um software gratuito, que provê toda a administração e a manutenção de banco de dados, como o controle dos privilégios de acesso, ajustes de performance, backup e recuperação.

O modelo relacional presente no PostgreSQL apresenta os dados como uma coleção tabelas, que possui uma estrutura que se repete a cada linha, como pode-se observar ao utilizar uma planilha, onde cada linha representa um registro e cada coluna, um tipo de dado.

Através do modelo relacional pode-se garantir a segurança dos registros inseridos no banco por meio da propriedade *ACID*, que significa, Automaticidade, Consistência, Integridade e Durabilidade dos dados.

O PostgreSQL é o *SGBD* utilizado no desenvolvimento do *Web Service Ambiental,* a fim de armazenar todos os dados que são buscados pela aplicação.

3.3 XML

Uma linguagem de marcação, cujo, é trabalhado de forma agregada ao código, podendo ser aplicados a dados ou textos a fim de serem lidos por computadores ou pessoas, sua finalidade é padronizar uma sequência de dados com o objetivo de organizar, separar o conteúdo e integrá-lo com outras linguagens.

3.4 JSON

É um protocolo leve para intercâmbio de dados e está baseado em um subconjunto da linguagem de programação *JavaScript,* sendo independente desta e de qualquer linguagem.

3.5 Framework Vraptor 4

Nada mais é do que um framework *MVC* (*Model, View, Controller*) web para desenvolvimento rápido de aplicações *Web*, cujo se utiliza de conceitos de inversão de controles, notações e injeções de dependências. Deste modo, foi utilizado a linguagem de programação Java para a construção do *Web Service*.

3.6 Maven

É uma ferramenta de integração de projetos ao qual é responsável por controlar versões de artefatos, gerar relatórios de produtividade, garantir execução de testes, manter nível de qualidade do código e gerenciamento de dependências, dentre outras funcionalidades.

4. ESTRUTURAÇÃO

Conforme o cronograma do projeto, a primeira etapa foi de análise e estudo da Linguagem de Programação Java Orientada a Objetos com integração ao banco de dados PostgreSQL, utilizando-se da interface gráfica pgAdmin. Em seguinda, foi atribuído conhecimentos sobre *Web Services* e a utilização do framework Vraptor 4, a fim de construir uma excelente aplicação, de modo que retorne os ultimos logs disponíveis ao usuário em formato *JSON*. A seguir é demonstrado a estrutura e o funcionamento do Web Service Ambiental.

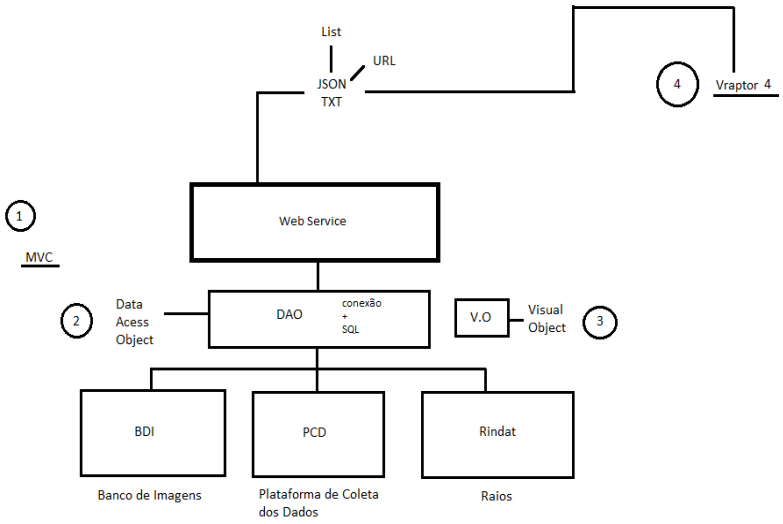

Figura 1: Estrutura Web Service Ambiental

Para a construção do projeto é utilizado a IDE Elipse, em conjunto com o arquivo .java, que representa a conexão com o banco de dados PostgreSQL para o desenvolvimento da aplicação foi utilizado também o padrão MVC (Model-View-Controller) seguindo o modelo orientação a objetos para a organização das classes.

A Figura 2 apresenta a base de dados pgAdmin e o banco BDI (Banco de Imagens) utilizado para construir esta aplicação do Web Service.

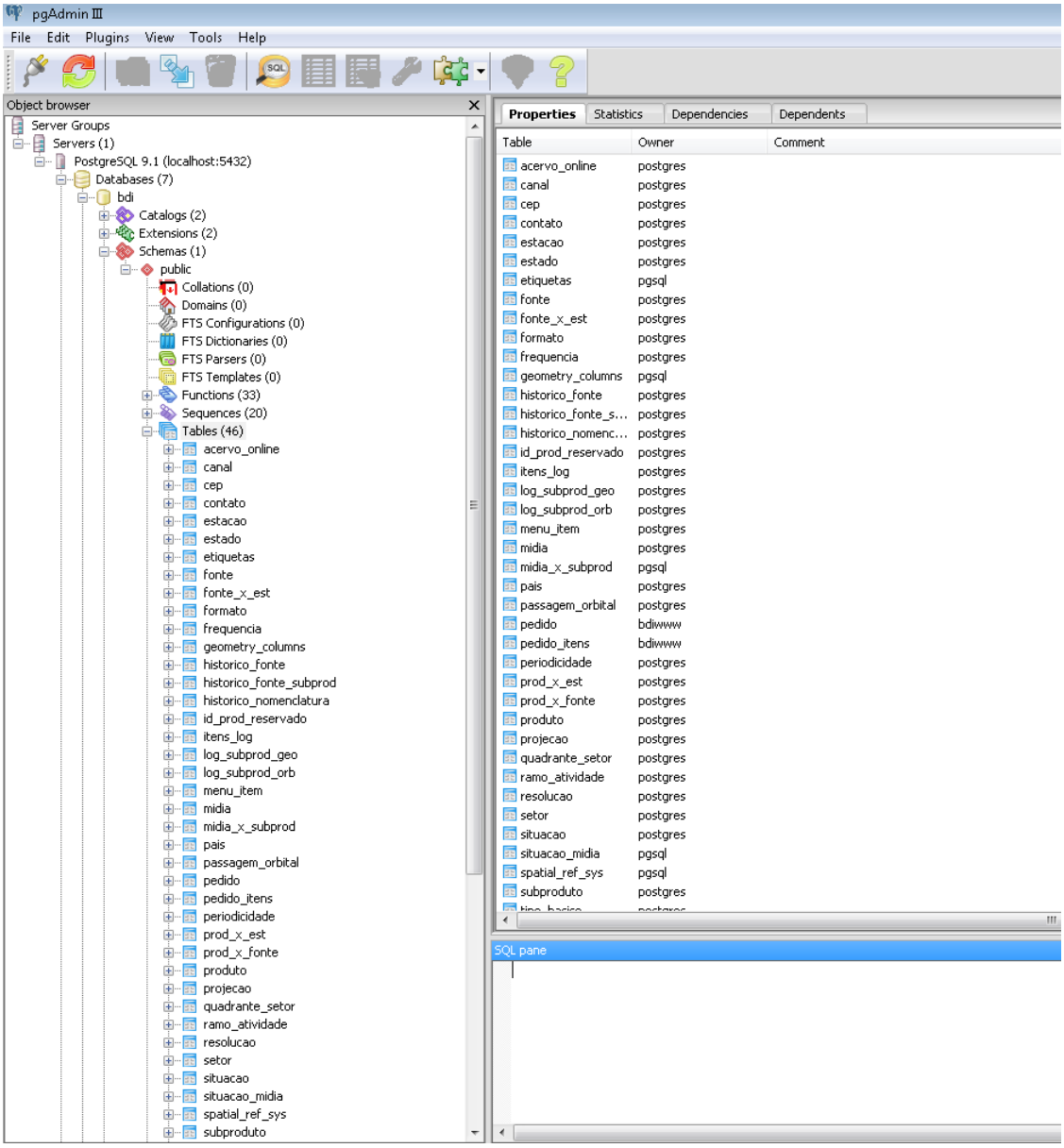

Figura 2: Interface pgAdmin

5. RESULTADOS

A Figura 3 apresenta o **pom.xml** sendo responsável por conter todas as dependências do projeto.

rest/pom.xml | Imagem.java | ImagemDao.java | ConnectionFactory.java | ImagemController.java

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com</groupId>
  <artifactId>rest</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>rest</name>

  <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>

        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <dependency>
            <groupId>javax.inject</groupId>
            <artifactId>javax.inject</artifactId>
            <version>1</version>
        </dependency>

        <dependency>
            <groupId>br.com.caelum</groupId>
            <artifactId>vraptor</artifactId>
            <version>4.1.0.Final</version>
        </dependency>

        <dependency>
            <groupId>org.jboss.weld.servlet</groupId>
            <artifactId>weld-servlet-core</artifactId>
            <version>2.1.2.Final</version>
        </dependency>

        <dependency>
            <groupId>org.jboss.weld</groupId>
            <artifactId>weld-core-impl</artifactId>
            <version>2.1.2.Final</version>
        </dependency>
```

Figura 3: Interface Eclipse – pom.xml

Em diante será demonstrado a criação das classes compostas no projeto de web service ambiental. A seguir exibe a Figura 4 da classe Imagem responsável pelos *setters* e *getters* da aplicação.

rest/pom.xml | Imagem.java | ImagemDao.java | ConnectionFactory.java | ImagemController.java

```
import java.util.ArrayList;

@Entity
public class Imagem {
    @Id // Define o id_usuario identificdor da tabela
    @SequenceGenerator (name = "imagem_seq", sequenceName = "imagem_seq", allocationSize = 1) // Forma de auto incremento de determinado campo
    @GeneratedValue (strategy = GenerationType.SEQUENCE, generator = "imagem_seq")
    private int id_log_subprod_orb;
    private String dt_subprod_orb;
    private String hora_subprod_orb;
    private String tam_arq;


    public Imagem() {}

    public Imagem(int id_log_subprod_orb, String dt_subprod_orb, String hora_subprod_orb, String tam_arq) {
         this.id_log_subprod_orb = id_log_subprod_orb;
         this.dt_subprod_orb = dt_subprod_orb;
         this.hora_subprod_orb = hora_subprod_orb;
         this.tam_arq = tam_arq;
    }
    public int getId_log_subprod_orb() {
        return id_log_subprod_orb;
    }
     public void setId_log_subprod_orb(int id_log_subprod_orb) {
        this.id_log_subprod_orb = id_log_subprod_orb;
    }
    public String getDt_subprod_orb() {
        return dt_subprod_orb;
    }
    public void setDt_subprod_orb(String dt_subprod_orb) {
        this.dt_subprod_orb = dt_subprod_orb;
    }
    public String getHora_subprod_orb() {
        return hora_subprod_orb;
    }
    public void setHora_subprod_orb(String hora_subprod_orb) {
        this.hora_subprod_orb = hora_subprod_orb;
    }
    public String getTam_arq() {
        return tam_arq;
    }
    public void setTam_arq(String tam_arq) {
        this.tam_arq = tam_arq;
    }
    public ArrayList<Imagem> getList() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Figura 4: Inteface Eclipse – classe Imagem

Para a Figura 5 é demonstrada a classe para acesso aos dados utilizando-se de query.

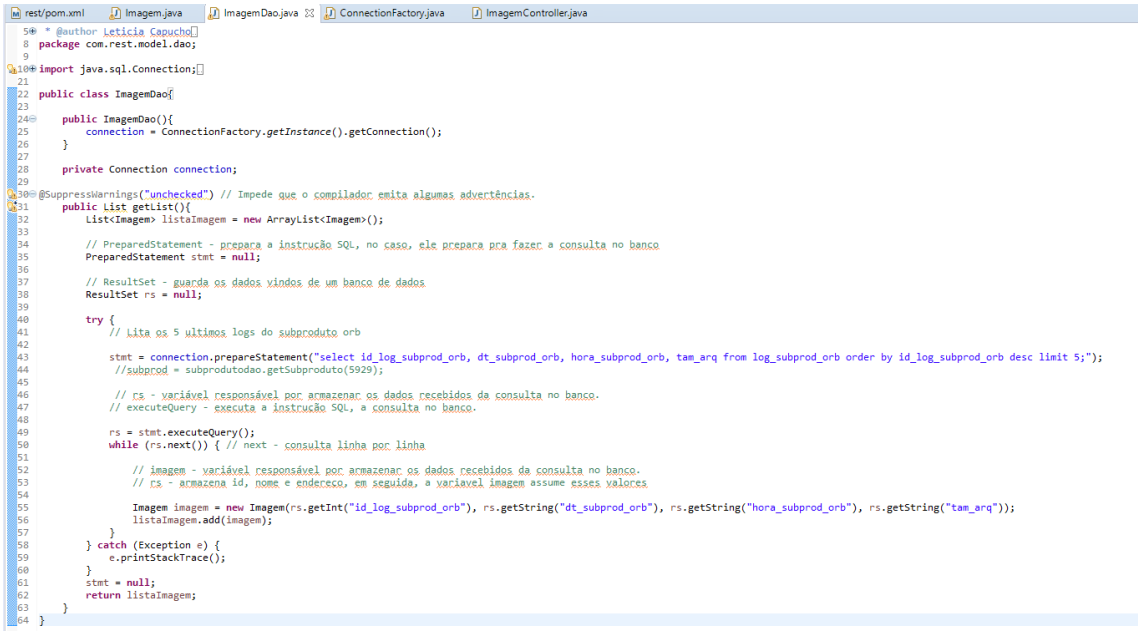

Figura 5: Inteface Eclipse – classe ImagemDao

Na Figura 6 apresenta a IDE de desenvolvimento adotada, Eclipse, em conjunto com o arquivo .java, que representa a conexão com o banco de dados PostGreSQL para o desenvolvimento da aplicação.

ConnectionFactory.java | ImagemController.java | ImagemDao.java | Imagem.java | beans.xml | hibernate.cfg.xml

```
5⊕ * @author Leticia Capucho
8  package com.rest.model.dao;
9
10⊕ import java.sql.Connection;
16
17 public class ConnectionFactory {
18     Connection con = null;
19     private static ConnectionFactory instance = null;
20
21⊖    public ConnectionFactory() {
22         inicio();
23     }
24⊖        public void inicio() {
25         try {
26             Class.forName("org.postgresql.Driver");
27             System.out.println("Driver Carregado");
28         } catch (ClassNotFoundException e) {
29             System.out.println("O driver do Postgre não pôde ser carregado!");
30         }
31     }
32⊖    public static ConnectionFactory getInstance() {
33         if (instance == null) {
34             instance = new ConnectionFactory();
35         }
36         return instance;
37     }
38⊖    public Connection getConnection() {
39         try {
40             return DriverManager.getConnection("jdbc:postgresql://pumina:5432/bdi", "postgres","1234");
41             } catch (SQLException e) {
42                 throw new RuntimeException(e);
43         }
44     }
45⊖    public void destroy(Connection conexao, PreparedStatement pstmt, ResultSet rs) {
46         try {
47             con.close();
48         } catch (SQLException e) {
49             System.out.println(e.getMessage());
50         }
51     }
52 }
```

Figura 6: Inteface Eclipse – classe ConnectionFactory

O método *getConnection* tem por objetivo fazer a conexão com o banco de dados do Postgre. Deste modo, dentro da condição try é especificado o nome do banco postgresql, o nome da máquina ao qual está sendo acessado esse banco, no caso, a “pumina” e a porta em que o servidor está configurado, que por padrão a porta TCP/IP (Protocolo de Controle de Transmissão/Protocolo de Internet) de conexão do PostgreSQL é 5432. Em seguida é identificado o nome no banco “bdi”, após esse caminho, é separado por vírgulas o usuário e a senha. Conforme exibe a Figura 7.

```
public Connection getConnection() {
    try {
        return DriverManager.getConnection("jdbc:postgresql://pumina:5432/bdi", "postgres","1234");
        } catch (SQLException e) {
            throw new RuntimeException(e);
    }
}
```

Figura 7: Inteface Eclipse – método Connection

Na Figura 10 foi utilizado a anotação @Controller a fim de estruturar a Controller no Vraptor. Já a anotação @Path é responsável de definir a url que será utilizada para o retorno da lista dos dados de banco., por padrão o VRaptor já assume /imagem ignorando o sufixo Controller do nome da classe ImagemController e listAll para o método listAll(). Concluíndo, para o retorno utilizou-se do Results.json()).

ConnectionFactory.java | ImagemController.java | ImagemDao.java | Imagem.java | beans.xml

```
 5⊕ * @author Leticia Capucho
 8 package com.rest.controller;
 9
10⊖ import javax.inject.Inject;
11
12 import com.rest.model.dao.ImagemDao;
13
14 import br.com.caelum.vraptor.Controller;
15 import br.com.caelum.vraptor.Get;
16 import br.com.caelum.vraptor.Path;
17 import br.com.caelum.vraptor.Result;
18 import br.com.caelum.vraptor.view.Results;
19
20 @Controller
21 public class ImagemController {
22⊖     @Inject // Permite a injestão de dependências na classe
23     private Result result;
24⊖     @Inject
25     private ImagemDao imagemDao;
26
27⊖     @Get
28     @Path("/imagem/list") //caminho para inserir na url
29     public void listAll() {
30
31       result.use(Results.json()).withoutRoot().from(imagemDao.getList()).serialize();
32     }
33 }
```

Figura 8: Inteface Eclipse – classe ImagemController

Como mencionado anteriormente, utilizou-se o Banco BDI (Banco de Dados Imagens), para consulta específica das últimas 5 imagens armazenadas, em ordem decrescente, conforme demonstrado na Figura 9

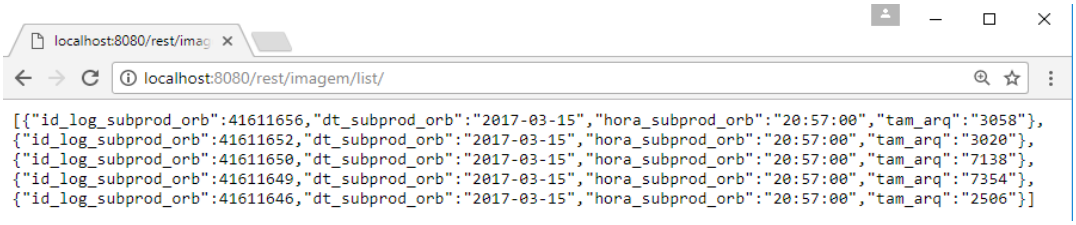

Figura 9: Inteface Web – retorno json

6. CONSIDERAÇÕES FINAIS

Diante da necessidade em oferecer uma camada de acesso eficiente para a disponibilização de dados ambientais e de imagens de satélite aos pesquisadores internos e externos ao CPTEC, foram avaliadas as tecnologias de *FTP* e *Web Service.* O *FTP (Protocolo de Transmissão de Arquivos)* trabalha no modo cliente/servidor, pois a transferência ocorre entre um computador "cliente", que solicita a conexão para a transferência de dados e o servidor, que recebe a solicitação de transferência. Para existir uma conexão com servidor, o cliente dever informar as credenciais de acesso, usuário e senha, sendo possível realizar a troca de dados. No entanto, foi observado que, este protocolo não atende bem o nosso modelo, tornando lento o processo de recebimento de dados, visto que, este depende da conexão do usuário para realizar o download. Com o intuito de diminuir o tempo de resposta aos dados de imagens, foi proposto a utilização de um *Web Service,* que é objeto desse trabalho, a fim de que as consultas requeridas pelos usuários sejam entregues de forma mais rápida e precisa. Com a utilização de um *Web Service* é possível obter uma solução na integração de sistemas e na comunicação entre aplicações distintas. Deste modo, para este projeto, foi fornecido uma camada padronizada e organizada de acesso e distribuição dos dados ambientais armazenados na DSA (Divisão de Satélite e Sistemas Ambientais), permitindo a integração desse conjunto de dados com os demais sistemas também fornecidos pela divisão. Para os usuários, foi proporcionado a disponibilização de um grande acervo armazenado na Divisão, que conta com Imagens de Satélite e Dados Ambientais (dados de Queimadas, Plataformas de Coleta, Raios e outros) que serão disponibilizados no formato JSON. No caso das imagens, utilizamos como base o endereço fornecido pelo BDI, fornecendo aos usuários o endereço da imagem (Metadado).

7. REFERÊNCIAS

Devmedia. **Artigo Java Magazine 61 – MVC Fácil com o Vraptor**. Disponível em: <http://www.devmedia.com.br/artigo-java-magazine-61-mvc-facil-com-o-vraptor/10522>. Acesso em: 13 de Junho de 2017.

Dclick. **O que é Maven, e seus primeiros passos com a ferramenta.** http://www.dclick.com.br/2010/09/15/o-que-e-o-maven-e-seus-primeiros-passos-com-a-ferramenta/>. Acesso em: 15 de Junho de 2017.

Macaratii. **JSON - Introdução e conceitos básicos.** Disponível em: <http://www.macoratti.net/13/07/net_json.htm>. Acesso em: 05 de Junho de 2017.

MESQUITA, Joao – **O QUE É O FTP E PARA QUE SERVE?** Disponível em: <http://domainer.pt/o-que-e-o-ftp-e-para-que-serve/>. Acesso em: 16 de Junho de 2017.

PEREIRA, Ana Paula. **O que é Java?** Disponível em: <https://www.tecmundo.com.br/programacao/2710-o-que-e-java-.htm Minas Gerais, Brasil>. Acesso em: 19 de Maio de 2017.

PEREIRA, Ana Paula. **O que é XML?** Disponível em: <https://www.tecmundo.com.br/programacao/1762-o-que-e-xml-.htm>. Acesso em: 18 de Maio de 2017.

SILVA, Pedro Henrique de Oliveira. **03 Aplicação Web com VRaptor 4 – Olá Vraptor.** Disponível em: <https://pedrohosilva.wordpress.com/2015/01/01/aplicacao-web-com-vraptor-4-pt3/>. Acesso em: 10 de Junho de 2017

SILVA, Pedro Henrique de Oliveira.**04 Aplicação Web com VRaptor 4 – Iniciando Projeto.** Disponível em: <https://pedrohosilva.wordpress.com/2015/01/03/aplicacao-web-com-vraptor-4-pt4/>. Acesso em: 19 de Junho de 2017.

SIÉCOLA, Paulo. **REST Construa API's inteligentes de maneira simples.** Disponível em: <https://www.casadocodigo.com.br/products/livro-rest>. Acesso em: 05 de Junho de 2017.

Vraptor. **Compartilhando objetos com a view**. Disponível em: <http://www.vraptor.org/pt/docs/trabalhando-com-a-view/>. Acesso em: 10 de Junho de 2017.