STD29006 – Sistemas Distribuídos 15/05/2019

Laboratório 4: Desenvolvendo Serviços Web em Java

Professor: Emerson Ribeiro de Mello http://docente.ifsc.edu.br/mello/std

Nota:

O objetivo deste laboratório é mostrar como desenvolver um Serviço Web RESTful em Java. O serviço será consumido por meio do aplicativo `curl` *e também por uma aplicação em Java. Requisitos:*

- *IntelliJ*
- *Apache Tomcat9*
- *JDK1.8*
- *curl*

Sumário

1 API Java para RESTful Web Services (JAX-RS)

A API Java para RESTful Web Services (JAX-RS) faz uso de Anotações Java para o desenvolvimento de clientes e serviços REST. O Jersey é a implementação de referência para a especificação JSR 311 e JAX-RS provê suporte para criação de mensagens XML e JSON por meio do do JAXB. A Tabela 1 apresenta as Anotações REST do JAX-RS.

Tabela 1: Anotações do JAX-RS

Anotação	**Descrição**
@PATH(caminho)	Define o caminho base da aplicação
@PUT	Indica que o método abaixo da anotação irá responder pedidos HTTP PUT
@GET	Indica que o método abaixo da anotação irá responder pedidos HTTP GET
@POST	Indica que o método abaixo da anotação irá responder pedidos HTTP POST
@DELETE	Indica que o método abaixo da anotação irá responder pedidos HTTP DELETE
@Produces(MediaType. TEXT_PLAIN[, outros tipos])	Define o tipo MIME que será entregue por um método que tiver sido anotado com @GET. Outros exemplos: “application/xml”, “application/json”
@Consumes(type[, outros tipos])	Define o tipo MIME que será consumido pelo método
@PathParam	Para obter valores de URL e inserir como parâmetro em um método

1.1 Algumas boas práticas para construção de API RESTful

- Sempre indique a versão da API. Combine a letra **v** com um número inteiro. Isso permitirá que você evolua sua aplicação sem quebrar a funcionalidade de aplicações de terceiros que foram construídas com uma versão específica de sua API.
  - Ex: `/api/v1/exemplo`, `/api/v2/novoExemplo`
- Prefira substantivos a verbos para nomear recursos. Prefira a escrita no plural
  - **Bons:** `/pessoas`, `/carros`, `/livros`
  - **Ruins:** `/obterPessoa`, `/criarPessoa`, `/apagarPessoa`
- Use os verbos HTTP para fazer operações criação (`POST`), recuperação (`GET`), atualização (`PUT`) e remoção (`DELETE`)
  - Requisições `GET` não devem alterar o estado da aplicação – requisição idempotente
- Faça uso de sub-recursos para indicar relacionamentos entre recursos
  - `GET /blocoA/labSiDi/computadores` – retorna a lista de computadores do laboratório LabSiDi que fica situado no bloco A
- Use o cabeçalho HTTP para indicar os formatos de serialização dos dados trocados
  - `Content-Type` indica o formato do pedido
  - `Accept` indica o formato da resposta
- Faça uso dos códigos de estado do HTTP nos retornos.
  - `200 OK`, `201 CREATED`, `404 NOT FOUND`, etc.

2 Estrutura do projeto para oferta de uma API RESTful

Nessa seção é demonstrado como criar a estrutura de diretórios e conjunto de arquivos que são base para qualquer projeto Java com a ferramenta *gradle* e usando a API JAX-RS.

2.1 Criando estrutura do projeto

Abaixo é demonstrado como criar estrutura do projeto usando somente o `gradle` na linha de comando e também é demonstrado como criar usando o IntelliJ.

2.1.1 Na linha de comando com o gradle

```
mkdir lab-rest-java
cd lab-rest-java

gradle init --type java-application

mkdir -p src/main/webapp/WEB-INF
touch src/main/webapp/WEB-INF/web.xml
touch src/main/webapp/index.jsp
```

Será gerada a seguinte estrutura de diretórios:

```
|-- build.gradle
|-- gradle
|   `-- wrapper
|       |-- gradle-wrapper.jar
|       `-- gradle-wrapper.properties
|-- gradlew
|-- gradlew.bat
|-- settings.gradle
`-- src
    |-- main
    |   |-- java
    |   |   `-- App.java
    |   `-- webapp
    |       |-- index.jsp
    |       `-- WEB-INF
    |           `-- web.xml
    `-- test
        `-- java
            `-- AppTest.java
```

2.1.2 Com o IntelliJ Ultimate 2019.1

1. Criar novo projeto **Gradle**, deixar marcado **Java** e **Web** em *Additional Libraries and Frameworks*. Escolher o `Java 1.8` para o **Project SDK**
   - Informe esses valores para **Groupid**, **ArtifactId** e **Version**: `engtelecom, std e 1.0`
   - Marque a opção **Use auto-import**
2. Em Project Name deixe: **lab-rest-java**
3. Criar o diretório `WEB-INF` dentro do diretório `src/main/webapp`
4. Criar o arquivo `web.xml` dentro do diretório **WEB-INF**
   - O IntelliJ apresentará uma notificação com o título **Frameworks Detected**. Clique no *link* **configure** e depois clique no **botão OK**.

2.2 Conteúdo do arquivo `build.gradle`

Edite o arquivo `build.gradle` que está na raiz do projeto e deixe seu conteúdo igual ao do quadro abaixo. Baixe o arquivo em: http://docente.ifsc.edu.br/mello/std/labs/lab-rest-java/build.gradle.

```
plugins {
  id 'java'
  id 'war'
}

apply from: 'https://raw.github.com/akhikhl/gretty/master/pluginScripts/gretty.plugin'

project.webAppDirName = 'src/main/webapp'

group 'engtelecom'
version '1.0'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

dependencies {
  testCompile group: 'junit', name: 'junit', version: '4.12'
  compile group: 'org.glassfish.jersey.containers', name: 'jersey-container-servlet', version: '
    2.28'
  compile group: 'org.glassfish.jersey.inject', name: 'jersey-hk2', version: '2.28'
  compile group: 'org.glassfish.jersey.media', name: 'jersey-media-json-jackson', version: '2.28'
  compile group: 'org.glassfish.jersey.media', name: 'jersey-media-multipart', version: '2.28'
  compile group: 'org.glassfish.jersey.media', name: 'jersey-media-json-processing', version: '
    2.28'
}

gretty {
  servletContainer = 'tomcat8'
  contextPath = '/'
}
```

2.3 Conteúdo do arquivo `web.xml`

Edite o arquivo `src/main/webapp/WEB-INF/web.xml` e deixe seu conteúdo igual ao do quadro abaixo. Baixe o arquivo em: http://docente.ifsc.edu.br/mello/std/labs/lab-rest-java/web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/
    javaee/web-app_3_1.xsd"
         version="3.1">
    <display-name>NomeDaMinhaApp</display-name>
    <servlet>
        <servlet-name>ExemploREST</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
      <!-- Atenção: Aqui deve colocar o pacote onde estão armazenadas as classes Java com
    serviços REST-->
            <param-value>engtelecom.std.rest</param-value>
        </init-param>
        <init-param>
            <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>ExemploREST</servlet-name>
        <url-pattern>/api/v1/*</url-pattern>
    </servlet-mapping>
</web-app>
```

2.4 Conteúdo do arquivo `index.jsp`

Edite o arquivo `src/main/webapp/index.jsp` e deixe seu conteúdo igual ao do quadro abaixo. Baixe o arquivo em: http://docente.ifsc.edu.br/mello/std/labs/lab-rest-java/index.jsp.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>Laboratório com serviços web RESTful em Java</title>
  </head>
  <body>
<h1>Laboratório com serviços web RESTful em Java</h1>

<H2>Primeiro exemplo</H2>

<ul>
    <li><a href="/api/v1/exemplo">Olá mundo</a></li>
    <li><a href="/api/v1/exemplo/Fulano">Olá Seu nome na URL</a></li>
    <li><a href="/api/v1/exemplo/pessoas">Dados de uma pessoa</a></li>
</ul>

<h2>Gestão de laboratórios</h2>
<ul>
    <li><a href="/api/v1/campi">Lista com todos os campi</a></li>
</ul>
</body>
</html>
```

3 Primeiro exemplo: Serviço para apresentação dos verbos HTTP

3.1 API do Serviço Web do Primeiro Exemplo

- **GET exemplo**
  - **Resultado:** a cadeia de caracteres “Olá mundo” em texto com a codificação UTF-8 e com código `HTTP 200 OK`
- **GET exemplo/{nome}**
  - **Resultado:** a cadeia de caracteres “Olá **nome**” em texto com a codificação UTF-8 e com código `HTTP 200 OK`
- **GET exemplo/pessoas**
  - **Resultado:** Retornará os dados de uma Pessoa em JSON ou XML
- **POST exemplo/pessoas**
  - **Resultado:** Receberá uma Pessoa representada em JSON e retornará o código `HTTP 201 CREATED` com a URI para o recurso dessa no campo *Location* do cabeçalho HTTP.
- **PUT exemplo/pessoas**
  - **Resultado:** Receberá uma Pessoa representada em JSON e retornará a cadeia de caracteres “Pessoa atualizada”
- **DELETE exemplo/pessoas/{pessoa}**
  - **Resultado:** Se o nome da pessoa for **Fulano**, então retornará a cadeia de caracteres “Pessoa excluída”, senão retornará erro com o código `HTTP 404 NOT FOUND`

3.2 Usando o `curl` para fazer requisições na API do Primeiro Exemplo

Assim que o serviço estiver em execução, faça uso das instruções abaixo para consumi-lo.

```
curl http://localhost:8080/api/v1/exemplo

curl http://localhost:8080/api/v1/exemplo/fulano

curl http://localhost:8080/api/v1/exemplo/pessoas

curl -X POST http://localhost:8080/api/v1/exemplo/pessoas -H "Content-Type: application/json"
    -d '{"nome": "Juca", "email":"novo@email.com"}'

curl -X PUT http://localhost:8080/api/v1/exemplo/pessoas -H "Content-Type: application/json"
    -d '{"nome": "Juca", "email":"novo@email.com"}'

curl -X DELETE http://localhost:8080/api/v1/exemplo/pessoas/fulano
```

Nota:

Existem diversas aplicações e plugins para que possa testar sua API REST. Por exemplo: ATOM[a], Visual Studio Code[b] e Postman[c].

[a] https://atom.io/
[b] https://code.visualstudio.com/
[c] https://www.getpostman.com/

3.3 Implementação do serviço

A classe `engtelecom.std.restPrimeiroExemplo.java` é onde estará implementado o serviço REST. Para esse auxiliar no exemplo foi criada o POJO `Pessoa.java`. A classe `Pessoa` só possui 2 atributos: `String nome` e `String email`. Crie os seguintes pacotes e classes dentro do diretório `src/main/java`.

```
`-- src
    |-- main
    |   |-- java
    |   |   `-- engtelecom
    |   |       `-- std
    |   |           |-- entities
    |   |           |   `-- Pessoa.java
    |   |           `-- rest
    |   |               `-- PrimeiroExemplo.java
```

Código 1: PrimeiroExemplo.java

```java
package engtelecom.std.rest;

import engtelecom.std.entities.Pessoa;
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("exemplo")
public class PrimeiroExemplo {

    @GET
    @Produces(MediaType.TEXT_PLAIN + ";charset=utf-8")
    public Response olaMundo() {
        String mensagem = "Olá mundo";
        return Response.ok(mensagem).build();
    }

    @Path("{nome}")
    @GET
    @Produces(MediaType.TEXT_PLAIN + ";charset=utf-8")
    public Response olaNome(@PathParam("nome") String nome) {
        String mensagem = "Olá " + nome;

        return Response.ok(mensagem).build();
    }

    @Path("pessoas")
    @GET
    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
    public Response obtemPessoa() {
        Pessoa p = new Pessoa("Fulano", "email@email.com");
        return Response.ok(p).build();
    }

    @Path("pessoas")
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response addPessoa(Pessoa p, @Context UriInfo uriInfo) {
        // Retornando o código HTTP 201 com a URL do recurso criado no campo Location
        // do cabeçalho HTTP
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(p.getNome());

        return Response.created(builder.build()).build();
    }
```

```
    @Path("pessoas")
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    public Response updatePessoa(Pessoa p) {
        return Response.ok("Pessoa atualizada").build();
    }

    @Path("pessoas/{pessoa}")
    @DELETE
    @Produces(MediaType.TEXT_PLAIN)
    public Response removePessoa(@PathParam("pessoa") String nome) {
        if (nome.equalsIgnoreCase("Fulano")) {
            return Response.ok("Pessoa excluída").build();
        }
        throw new NotFoundException();
    }
}
```

O JAXB 2.0 permite mapear classes Java em XML ou JSON. Para isto as classes Java devem marcadas com Anotações Java `@XmlRootElement`. A classe `Pessoa` abaixo tem essa anotação.

```
package engtelecom.std.entities;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Pessoa {
    private String nome;
    private String email;

    public Pessoa(){}

    public Pessoa(String nome, String email) {
        this.nome = nome;
        this.email = email;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Nome: " + this.nome + ", Email: " + this.email;
    }
}
```

3.4 Colocando o serviço em execução com o `gretty`

O plugin `gretty`[1] para `gradle` permite a execução de aplicações *web* dentro de *servlet containers* embarcados (tomcat ou jetty) no próprio projeto. Ou seja, cria-se aqui um servidor de aplicação específico para o projeto. O `gretty` facilita a implantação da aplicação, início e parada do servidor.

3.4.1 Usando o gradle por linha de comando

Estando na raiz do diretório do projeto, execute os comandos:

```
gradle build

gradle appRun
```

3.4.2 Usando o gradle com o IntelliJ

Abra o painel do gradle (geralmente encontra-se na barra lateral direita).

- **Para compilar:** `Tasks` → `build` → `build`
- **Para executar:** `Tasks` → `gretty` → `appRun`

4 Cliente RESTful em Java

Crie um novo projeto Java com o *gradle* e deixe o conteúdo do arquivo `build.gradle` igual ao quadro abaixo:

```
plugins {
    id 'java'
}

group 'engtelecom'
version '1.0'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
    compile group: 'org.glassfish.jersey.containers', name: 'jersey-container-servlet', version:
     '2.28'
    compile group: 'org.glassfish.jersey.media', name: 'jersey-media-json-jackson', version: '
    2.28'
    compile group: 'org.glassfish.jersey.inject', name: 'jersey-hk2', version: '2.28'
}
```

- Crie um POJO `Pessoa.java` com os seguintes atributos: `String nome` e `String email`;
- Crie uma classe `Principal.java` e que possua um método `main` com o conteúdo da listagem abaixo:

[1] https://github.com/akhikhl/gretty

```
package std;

import org.glassfish.jersey.client.ClientConfig;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;

public class Principal {

    private ClientConfig config;
    private Client client;
    private WebTarget target;
    private final String URI = "http://localhost:8080/api/v1";

    public Principal(){
        this.config  = new ClientConfig();
        this.client = ClientBuilder.newClient(config);
        this.target = client.target(UriBuilder.fromUri(URI).build());
    }

    public void invocandoOlaMundo(){
        String jsonResponse = target.path("exemplo").request().accept(MediaType.TEXT_PLAIN).
    get(String.class);

        System.out.println("Resultado: " + jsonResponse);
    }

    public void adicionandoUmaPessoa(){
        Pessoa p = new Pessoa("fulano", "meu@email.com");

        Response response = target.path("exemplo").path("pessoas").request(MediaType.
    APPLICATION_JSON).post(Entity.entity(p,MediaType.APPLICATION_JSON), Response.class);

        System.out.println("Resultado: " + response.getStatus());
    }

    public static void main(String[] args) {
        Principal p = new Principal();

        p.invocandoOlaMundo();
        p.adicionandoUmaPessoa();
    }
}
```

5 Implantação da aplicação em um servidor de aplicação Apache Tomcat8

Os passos descritos na Subseção 3.4 são adequados para o ambiente de desenvolvimento, porém uma vez que a solução estiver pronta, é necessário implantar a mesma em um servidor de aplicação de produção. Nessa seção é apresentado como gerar o **arquivo .war** com o código aqui desenvolvido e como implantar o mesmo em uma instalação do Apache Tomcat8.

5.1 Implantação do arquivo `.war` no Apache Tomcat8

- Baixe o `.tar.gz` do Tomcat8 em http://tomcat.apache.org/ e descompacte o mesmo em seu computador
- Gere o arquivo `.war` de sua aplicação com o comando: `gradle war` (ou via painel Gradle no IntelliJ)
  - Será gerado o arquivo `build/libs/std-1.0.war`, copie o mesmo para o diretório `webapps/` dentro da instalação do Apache Tomcat8
- Coloque o Tomcat8 em execução
  - Defina a variável de ambiente `JRE_HOME` com o caminho da instalação do JDK1.8 e execute o *script* `bin/startup.sh` do Apache Tomcat8. Por exemplo:

```
export JRE_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64/

./bin/startup.sh
```

  - Acesse o serviço por meio da URL: http://localhost:8080/std-1.0/api/v1/exemplo

Nota:

O Apache Tomcat possui uma aplicação web, chamada Manager, que permite gerenciar e implantar aplicações (arquivos `.war`). Por padrão, essa aplicação só pode ser acessada via `localhost` pelo no endereço http:// localhost:8080/ manager. Porém, é necessário adicionar um usuário com permissões para usá-la. Edite o arquivo `conf/tomcat-users.xml` e adicione a seguinte linha dentro do elemento `<tomcat-users>`

```
<user username="administrador" password="supersenha" roles="tomcat,manager-gui"/>
```

6 Segundo exemplo: Serviço para gestão de laboratórios

Serviço para gestão dos laboratórios de ensino dos campi do IFSC.

6.1 API do Serviço Web para gestão de laboratórios – `api/v1`

- **GET campi** – Retorna um JSON com uma lista de todos os campi da instituição
- **GET campi/{sigla-campus}** – Retorna um JSON com detalhes de um campus
  - Nome do campus, sigla, telefone
- **GET campi/{sigla-campus}/blocos** – Retorna um JSON com uma lista de todos os blocos do campus
- **GET campi/{sigla-campus}/blocos/{bloco}/laboratorios** – Retorna um JSON com uma lista de todos os laboratórios do bloco
- **GET campi/{sigla-campus}/blocos/{bloco}/laboratorios/{sigla-laboratorio}** – Retorna um JSON com detalhes de um laboratório
  - Nome, Sigla, total de lugares, tipo do laboratório (hardware ou software)
- **POST campi** – Para adicionar um novo campus
  - **Parâmetros:** (em JSON)
    - nome
    - sigla
    - telefone
- **POST campi/{sigla-campus}/blocos** – Para adicionar um novo bloco
  - **Parâmetros:** (em JSON)
    - sigla
    - descricao
- **POST campi/{sigla-campus}/blocos/{bloco}/laboratorios** – Para adicionar um novo laboratório
  - **Parâmetros:** (em JSON)
    - nome
    - sigla
    - totalDeLugares
    - tipo – valores permitidos: hardware ou software
- **PUT campi/{sigla-campus}/blocos/{bloco}/laboratorios/{sigla-laboratorio}** – Para alterar dados de um laboratório
  - **Parâmetros:** (em JSON)
    - nome
    - sigla
    - totalDeLugares
    - tipo – valores permitidos: hardware ou software
- **DELETE campi/{sigla-campus}/blocos/{bloco}/laboratorios/{sigla-laboratorio}** – Para remover um laboratório por meio de sua sigla

6.2 Criar pacotes e classes Java

Criar os seguintes pacotes e classes dentro do diretório `src/main/java`:

```
`-- src
    |-- main
    |   |-- java
    |   |   `-- engtelecom
    |   |       `-- std
    |   |           |-- TipoLaboratorio.java  # esse arquivo é um enum e não uma classe
    |   |           |-- db
    |   |           |   |-- BancoDeDados.java # para persistir um banco em memória
    |   |           |   `-- CampiDAO.java     # para acessar o banco
    |   |           |-- entities
    |   |           |   |-- Bloco.java
    |   |           |   |-- Campus.java
    |   |           |   `-- Laboratorio.java
    |   |           `-- rest
    |   |               `-- Campi.java        # API do Serviço RESTful
```

Em http://docente.ifsc.edu.br/mello/std/labs/lab-rest-java/std/ estão disponíveis os códigos fontes das listagens abaixo.

Código 2: BancoDeDados.java

```java
package engtelecom.std.db;
public class BancoDeDados {

    private static BancoDeDados instance;
    public CampiDAO campiDAO;

    public BancoDeDados() {
        this.campiDAO = new CampiDAO();
    }

    public static synchronized BancoDeDados getInstance(){
        if (instance == null){
            instance = new BancoDeDados();
        }
        return instance;
    }
}
```

Código 3: CampiDAO.java

```java
package engtelecom.std.db;

import engtelecom.std.entities.Campus;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.HashMap;
import java.util.Map;

@XmlRootElement
public class CampiDAO {
    @XmlElement(name="campi")
    public Map<String, Campus> listaDosCampi;

    public CampiDAO() {
        this.listaDosCampi = new HashMap<>();
    }

    public Campus buscaCampusPelaSigla(String sigla){
        return listaDosCampi.get(sigla);
    }
}
```

Código 4: TipoLaboratorio.java

```
package engtelecom.std;

public enum TipoLaboratorio {
    HARDWARE("hardware"),
    SOFTWARE("software");

    private String descricao;

    private TipoLaboratorio(String desc){
        this.descricao = desc;
    }

    public String getDescricao() {
        return descricao;
    }
}
```

Código 5: Bloco.java

```
package engtelecom.std.entities;

import javax.xml.bind.annotation.XmlRootElement;
import java.util.HashMap;
import java.util.Map;

@XmlRootElement
public class Bloco {
    private String sigla;
    private String descricao;
    private Map<String,Laboratorio> laboratorios;

    public Bloco() {
        this.laboratorios = new HashMap<>();
    }

    public Bloco(String sigla, String descricao) {
        this.sigla = sigla;
        this.descricao = descricao;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public Laboratorio buscaLabPelaSigla(String lab){
        return this.laboratorios.get(lab);
    }

    public Map<String, Laboratorio> getLaboratorios() {
        return laboratorios;
    }
```

```
    public void setLaboratorios(Map<String, Laboratorio> laboratorios) {
        this.laboratorios = laboratorios;
    }
}
```

Código 6: Campus.java

```
package engtelecom.std.entities;

import javax.xml.bind.annotation.XmlRootElement;
import java.util.HashMap;
import java.util.Map;

@XmlRootElement
public class Campus {
    private String nome;
    private String sigla;
    private String telefone;
    private Map<String, Bloco> blocos;

    public Campus() {
        this.blocos = new HashMap<>();
    }

    public Campus(String nome, String sigla, String telefone) {
        super();
        this.nome = nome;
        this.sigla = sigla;
        this.telefone = telefone;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
    public Bloco buscaBlocoPelaSigla(String sigla){
       return this.blocos.get(sigla);
    }
    public Map<String, Bloco> getBlocos() {
        return blocos;
    }
    public void setBlocos(Map<String, Bloco> blocos) {
        this.blocos = blocos;
    }
}
```

Código 7: Laboratorio.java

```
package engtelecom.std.entities;

import engtelecom.std.TipoLaboratorio;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Laboratorio {
    private String nome;
    private String sigla;
    private int totalDeLugares;
    private TipoLaboratorio tipo;

    public Laboratorio() {
    }

    public Laboratorio(String nome, String sigla, int totalDeLugares, TipoLaboratorio tipo) {
        this.nome = nome;
        this.sigla = sigla;
        this.totalDeLugares = totalDeLugares;
        this.tipo = tipo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public int getTotalDeLugares() {
        return totalDeLugares;
    }

    public void setTotalDeLugares(int totalDeLugares) {
        this.totalDeLugares = totalDeLugares;
    }

    public TipoLaboratorio getTipo() {
        return tipo;
    }

    public void setTipo(TipoLaboratorio tipo) {
        this.tipo = tipo;
    }
}
```

6.3 Implementação do serviço

Código 8: Campi.java

```
package engtelecom.std.rest;

import engtelecom.std.db.BancoDeDados;
import engtelecom.std.db.CampiDAO;
import engtelecom.std.entities.Bloco;
import engtelecom.std.entities.Campus;
import engtelecom.std.entities.Laboratorio;

import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.util.Map;


@Path("campi")
public class Campi {

    private BancoDeDados bancoDeDados = BancoDeDados.getInstance();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public CampiDAO getListaCampi() {
        return bancoDeDados.campiDAO;
    }

    @Path("/{sigla-campus}")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getCampus(@PathParam("sigla-campus") String campus) {
        Campus c = bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase());
        if (c != null) {
            return Response.ok(c).build();
        }
        throw new NotFoundException();

    }

    @Path("/{sigla-campus}/blocos")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getListaBlocos(@PathParam("sigla-campus") String campus) {
        Campus c = bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase());
        if (c != null) {
            return Response.ok(c.getBlocos()).build();
        }
        throw new NotFoundException();
    }

    @Path("/{sigla-campus}/blocos/{bloco}/laboratorios")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getLabsDoBlocos(@PathParam("sigla-campus") String campus, @PathParam("
    bloco") String bloco) {
        try {
            Map<String, Laboratorio> m = this.bancoDeDados.campiDAO.buscaCampusPelaSigla(
    campus.toUpperCase()).buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios();
            return Response.ok(m).build();
        } catch (Exception e) {
            throw new NotFoundException();
        }
    }

    @Path("/{sigla-campus}/blocos/{bloco}/laboratorios/{sigla-lab}")
```

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getLabsDoBlocos(@PathParam("sigla-campus") String campus, @PathParam("
    bloco") String bloco, @PathParam("sigla-lab") String laboratorio) {

        try {
            Laboratorio lab = this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.
    toUpperCase()).buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios().get(laboratorio.
    toUpperCase());
            return Response.ok(lab).build();
        } catch (Exception e) {
            throw new NotFoundException();
        }


    }


    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    public Response addCampus(Campus c, @Context UriInfo uriInfo) {
        if (this.bancoDeDados.campiDAO.listaDosCampi.put(c.getSigla().toUpperCase(), c) ==
    null) {
            throw new InternalServerErrorException();
        }

        // Retornando o código HTTP 201 com a URL do recurso criado no campo Location
        //  do cabeçalho HTTP
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(c.getSigla());

        return Response.created(builder.build()).build();
    }

    @Path("/{sigla-campus}/blocos")
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    public Response addBloco(@PathParam("sigla-campus") String campus, Bloco b, @Context
    UriInfo uriInfo) {
        Campus c = this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase());
        if (c == null) {
            throw new NotFoundException();
        }
        c.getBlocos().put(b.getSigla().toUpperCase(), b);


        // Retornando o código HTTP 201 com a URL do recurso criado no campo Location
        // do cabeçalho HTTP
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(b.getSigla());

        return Response.created(builder.build()).build();
    }


    @Path("/{sigla-campus}/blocos/{bloco}/laboratorios")
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    public Response addLaboratorio(@PathParam("sigla-campus") String campus, @PathParam("bloco
    ") String bloco, Laboratorio lab, @Context UriInfo uriInfo) {

        try {
```

```
            this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase()).
    buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios().put(lab.getSigla().toUpperCase
    (), lab);
        } catch (Exception e) {
            throw new NotFoundException();
        }

        // Retornando o código HTTP 201 com a URL do recurso criado no campo Location
        // do cabeçalho HTTP
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(lab.getSigla());

        return Response.created(builder.build()).build();
    }

    @Path("/{sigla-campus}/blocos/{bloco}/laboratorios/{siglaLab}")
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces({MediaType.APPLICATION_JSON, MediaType.TEXT_PLAIN})
    public Response updateLaboratorio(@PathParam("sigla-campus") String campus, @PathParam("
    bloco") String bloco, @PathParam("siglaLab") String siglaLab, Laboratorio lab) {
        Laboratorio l;
        try {
            l = this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase()).
    buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios().get(siglaLab);

        } catch (Exception e) {
            throw new NotFoundException();
        }

        if (l != null) {
            this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase()).
    buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios().put(lab.getSigla().toUpperCase
    (), lab);
        }

        return Response.ok("sucesso").build();
    }

    @Path("/{sigla-campus}/blocos/{bloco}/laboratorios/{sigla-lab}")
    @DELETE
    @Produces(MediaType.TEXT_PLAIN)
    public Response removeLab(@PathParam("sigla-campus") String campus, @PathParam("bloco")
    String bloco, @PathParam("sigla-lab") String laboratorio) {
        Laboratorio l = this.bancoDeDados.campiDAO.buscaCampusPelaSigla(campus.toUpperCase()).
    buscaBlocoPelaSigla(bloco.toUpperCase()).getLaboratorios().remove(laboratorio.toUpperCase
    ());

        if (l != null) {
            return Response.ok("sucesso").build();
        }
        throw new NotFoundException();
        // Disparar a exceção acima seria equivalente ao retorno abaixo.
//        return Response.status(Response.Status.NOT_FOUND).entity("campus e/ou bloco e/ou
    laboratório não encontrado").build();
    }
}
```

Referências

1. https://guides.gradle.org/building-java-applications/
2. https://restfulapi.net/

3. http://www.vogella.com/tutorials/REST/article.html
4. https://dennis-xlc.gitbooks.io/restful-java-with-jax-rs-2-0-en/content/en/index.html