

FIAP GRADUAÇÃO

# DIGITAL BUSINESS ENABLEMENT

Prof. MSc. Rafael Matsuyama

#06 – WEB SERVICES RESTFUL



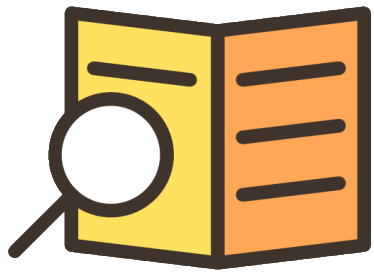
# PERCURSO

---



## #06 - AGENDA

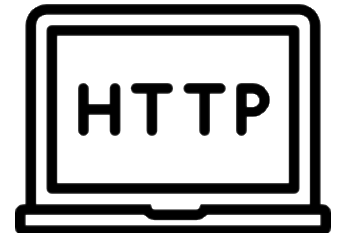
---



- **Web Services Restful**
  - URI, Métodos e Status Code do HTTP
- **Web Services Restful com Java**
  - JAS-RS, principais anotações
  - Criação do projeto e configuração
  - Implementação do CRUD com Json
- **Consumindo um Web Services Restful**
  - Criando e configurando o projeto
  - Consumindo o CRUD

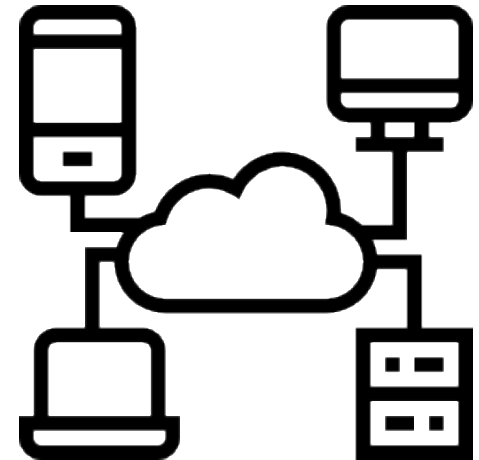
## RESTFul (REpresentational State Transfer)

- Simples, leve, fácil de desenvolver e evoluir;
- Tudo é um recurso (**Resource**);
- Cada recurso possui um identificador (**URI**);
- Recursos podem utilizar vários formatos: html, xml, **Json**;
- Utiliza o Protocolo **HTTP**;
- Os métodos HTTP: **GET, POST, PUT, DELETE** são utilizados na arquitetura REST.



O protocolo **HTTP** possui vários métodos, os principais:

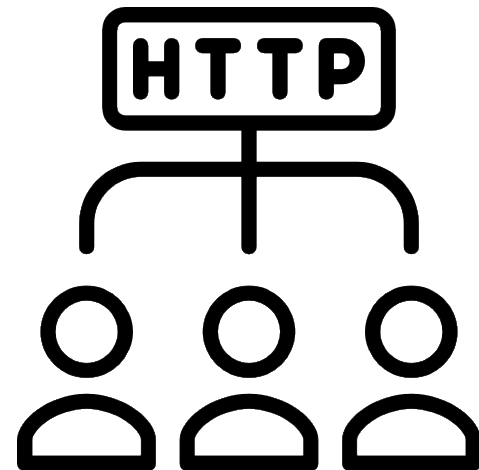
- **GET:** recupera informações de um recurso;
- **POST:** cria um novo recurso;
- **PUT:** atualiza um recurso;
- **DELETE:** remove um recurso;



- Quando realizamos uma requisição, é preciso determinar o **endereço** do **recurso** que vamos acessar:

VERBO	URI	AÇÃO
POST	/pedido/	Criar
GET	/pedido/1	Visualizar
PUT	/pedido/1	Alterar
DELETE	/pedido/1	Apagar

- Os códigos de **status das respostas HTTP** indicam se uma requisição HTTP foi corretamente concluída. As respostas são agrupadas em cinco classes: **respostas de informação, respostas de sucesso, redirecionamentos, erros do cliente e erros do servidor**;
- **1xx** – Informativa;
- **2xx** – Sucesso;
- **3xx** – Redirecionamentos;
- **4xx** – Erros do cliente;
- **5xx** – Erros do servidor;





- Vamos trabalhar com alguns códigos na implementação do **Web Service**:

CODE	DESCRIÇÃO
200	Ok
201	Created (Criado)
204	No Content (Sem conteúdo)
500	Internal Server Error
404	Not Found
405	Method not Allowed



# WEB SERVICE RESTFUL COM JAVA

## JAX-RS

- Especificação Java para suporte a REST (JSR 331);
- JAX-RS: Java API for RESTful Web Services;
- **Jersey**: implementação da especificação.



## Principais anotações:

Anotação	Descrição
@Path	Define o caminho para o recurso (URI).
@POST	Responde por requisições POST.
@GET	Responde por requisições GET.
@PUT	Responde por requisições PUT.
@DELETE	Responde por requisições DELETE.
@Produces	Define o tipo de informação que o recurso retorna.
@Consumes	Define o tipo de informação que o recurso recebe.
@PathParam	Injeta um parâmetro da URL no parâmetro do método.

# CRIANDO O PROJETO

**New Dynamic Web Project**

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v9.0 runtime. Additional facets can later be installed to add new functionality to the project.

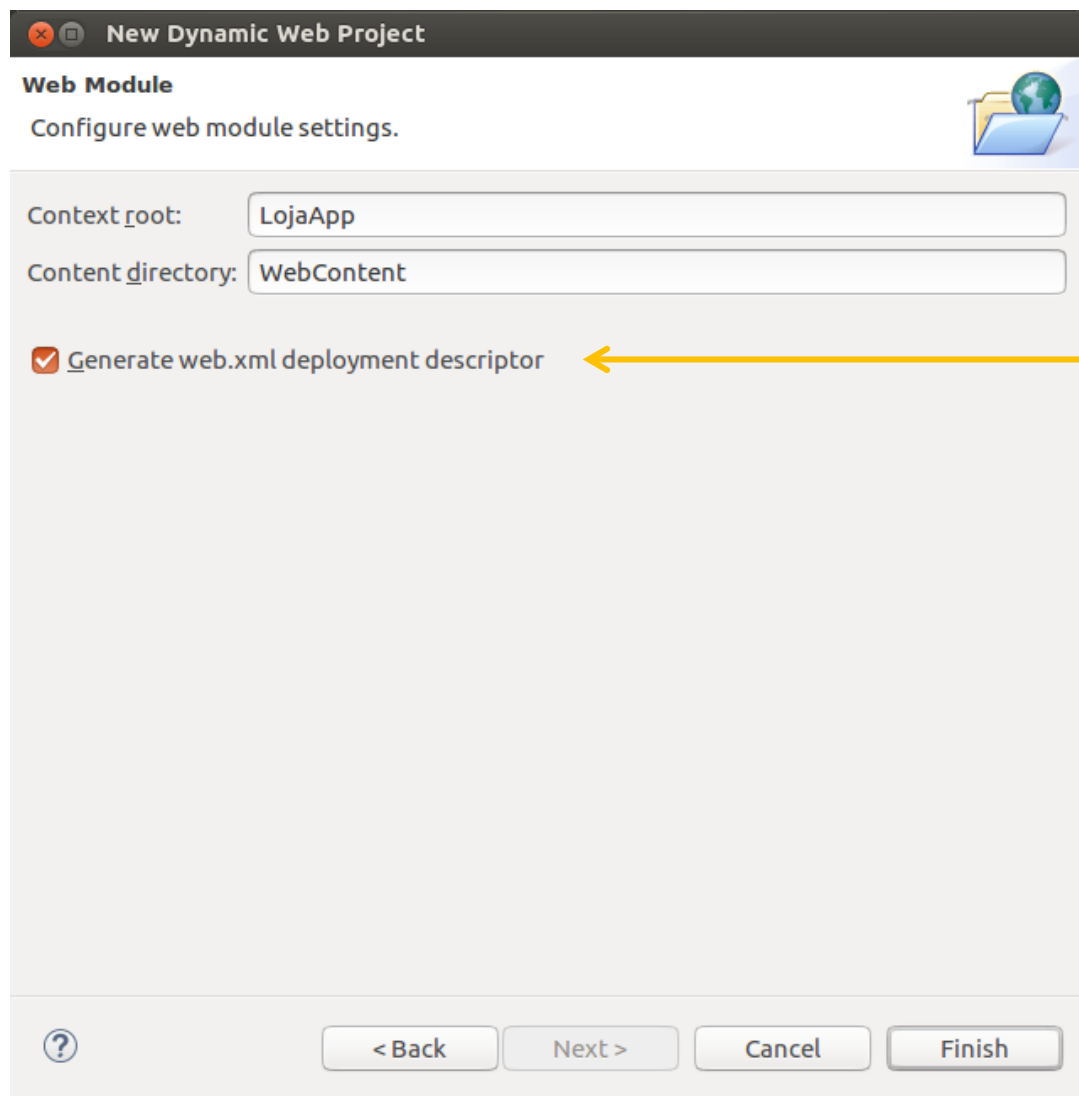
EAR membership

☐ Add project to an EAR

Crie um Dynamic Web Project com Tomcat e **web.xml**

Module Version 3.1

Next para gerar o web.xml



**New Dynamic Web Project**

**Web Module**  
Configure web module settings.

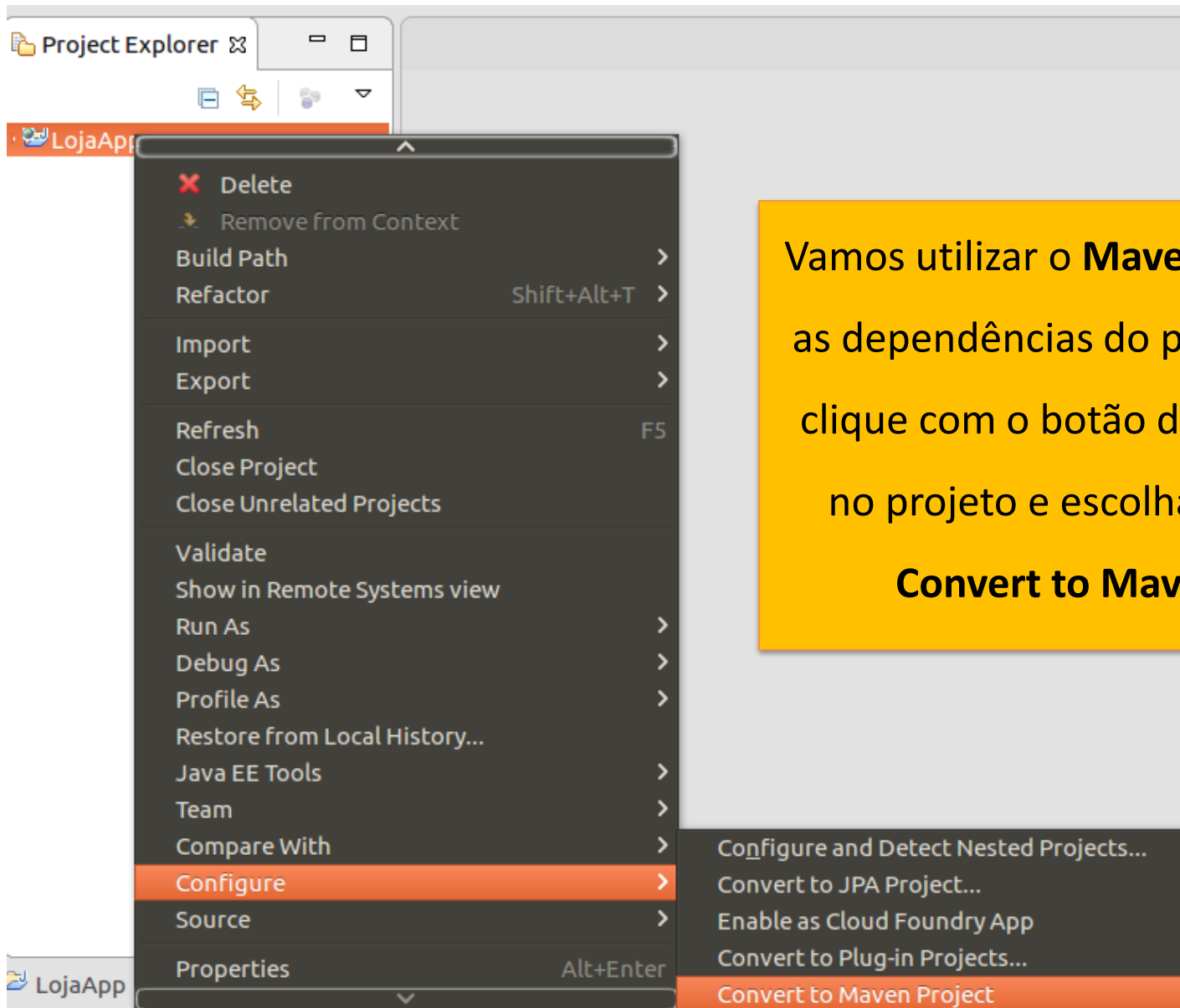
Context root: LojaApp

Content directory: WebContent

☒ Generate web.xml deployment descriptor

? < Back Next > Cancel Finish

Marque para gerar o  
**web.xml**



Vamos utilizar o **Maven** para gerenciar as dependências do projeto, para isso clique com o botão direito do mouse no projeto e escolha **Configure** → **Convert to Maven Project**

- **Maven** é uma ferramenta para o gerenciamento, construção e implantação de projetos Java. Com ele é possível gerenciar as dependências, o build e documentação.
- O arquivo **pom.xml** deve ficar na raiz do projeto e nele se declara a estrutura, dependências e características do seu projeto.

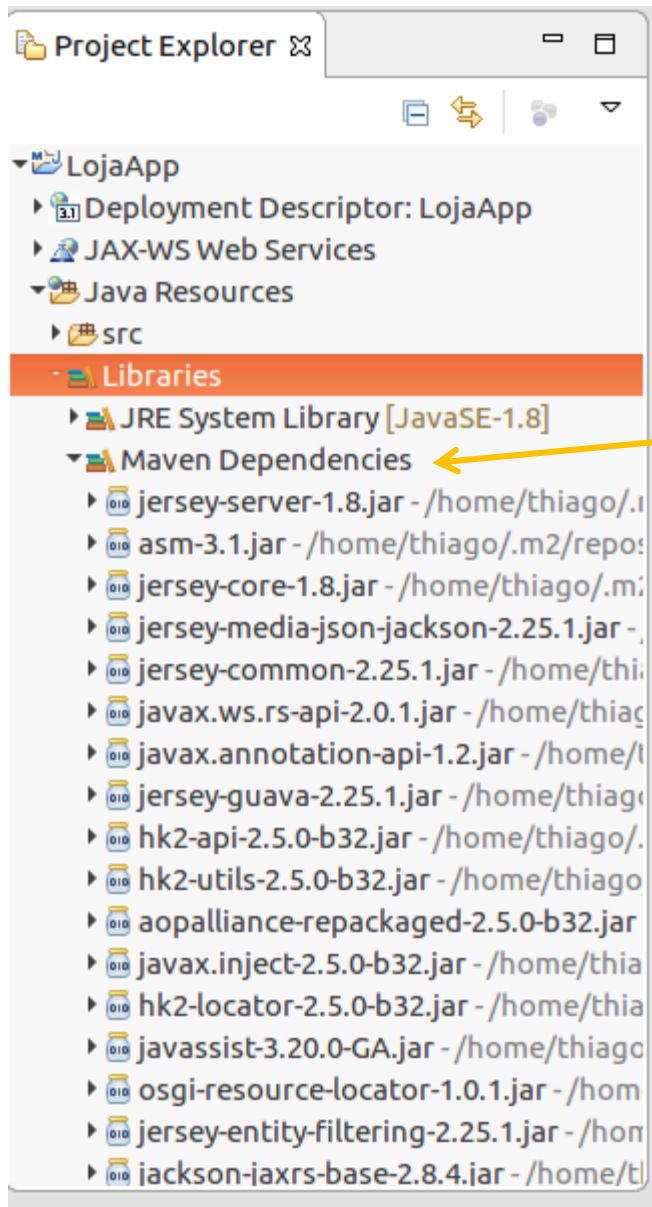




- Vamos configurar o **pom.xml** para adicionar as dependências do projeto:

```
<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-server</artifactId>
    <version>2.17</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet-core</artifactId>
    <version>2.17</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>2.25.1</version>
  </dependency>
</dependencies>
```

Adicione as **dependências** após a  
tag **</build>**



Após a configuração clique com o botão direito do mouse no projeto e escolha **Maven** →

**Update Project**

Depois é possível ver as **bibliotecas** (jar) que foram adicionadas ao projeto.

- Agora é preciso configurar o projeto para o **Restful**, no arquivo **web.xml** adicione:

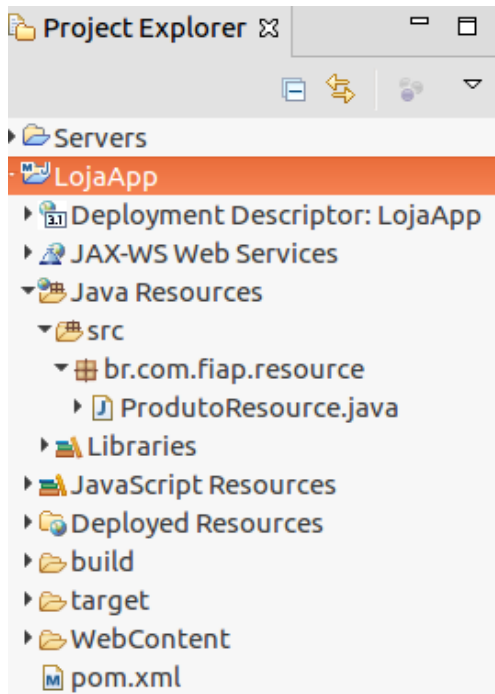
```
<servlet>
    <servlet-name>jersey-servlet</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>br.com.fiap.resource</param-value>
    </init-param>
    <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>jersey-servlet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Pacote onde estão as classes do web services

Parte da URL para acessar o web service

- Vamos criar uma classe **Java** no pacote configurado no **web.xml** chamada **ProdutoResource**, onde será desenvolvido os serviços.



```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/produto")
public class ProdutoResource {

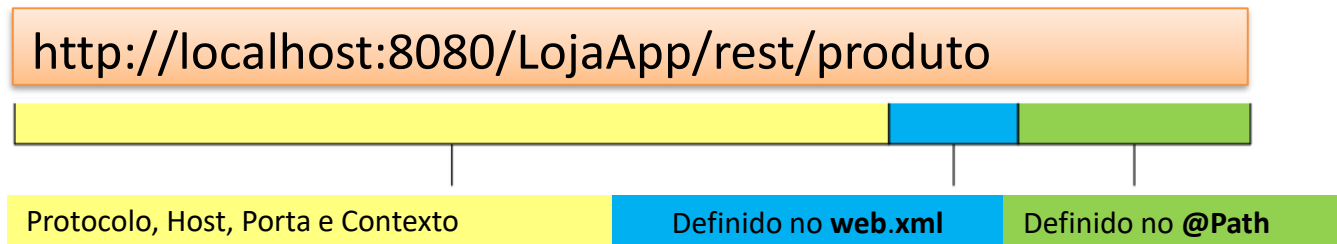
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String buscar(){
        return "Ola Mundo!";
    }

}
```

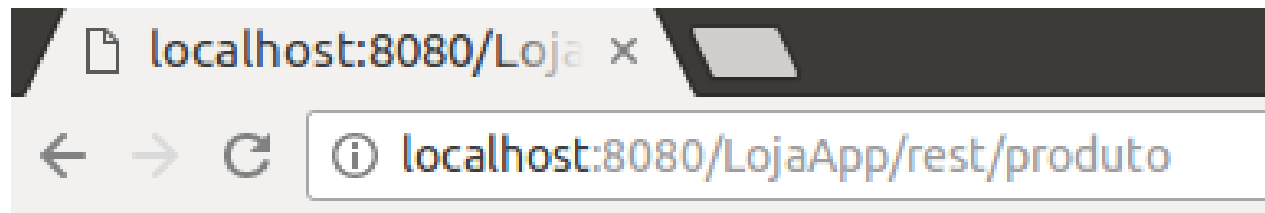
Parte do endereço da **URL** para acessar o serviço:

Requisições **GET** e retorna um texto puro como resposta.

- Execute o Servidor e faça uma chamada ao serviço através de sua **URL** no navegador:



## Navegador:

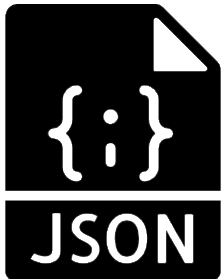


Ola Mundo!

- Ao invés de trabalhar com o envio de texto puro, é melhor trabalhar com **Json – JavaScript Object Notation**;
- Formato simples e leve para transferência de dados;
- Uma alternativa para o XML;

**Exemplo:**

```
{  
  "show": "Oasis",  
  "preco": 150,  
  "local": "São Paulo"  
}
```



Validador de formato Json: <http://jsonlint.com/>

## Exemplo de um array em JSON:

```
{  
  "shows": [  
    {  
      "show": "Oasis",  
      "preco": 150,  
      "local": "São Paulo"  
    },  
    {  
      "show": "Link Park",  
      "preco": 250,  
      "local": "Rio de Janeiro"  
    },  
    {  
      "show": "Jorge e Mateus",  
      "preco": 200,  
      "local": "São Paulo"  
    }  
  ]  
}
```

{ JSON }

Os colchetes [ ] limitam o array.



# GET - BUSCAS



- Vamos trabalhar com a biblioteca **Jackson** para converter objetos **Java** em representações **Json** e vice-versa.
- A **dependência** já foi adicionada no projeto, dessa forma a biblioteca irá realizar a **conversão automaticamente**.

```
@XmlElement
public class ProdutoTO {

    private int codigo;

    private String titulo;

    private double preco;

    private int quantidade;

    //construtores, gets e sets;

}
```

Anotação para identificar que a classe pode ser transformada em Json;

Crie a classe para **armazenar** as informações do **produto**.

# LISTAR TODOS OS PRODUTOS

- Vamos ajustar o código do serviço para que ele retorne **todos** os **produtos** cadastrados!

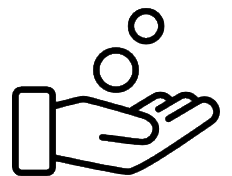
```
@Path("/produto")
public class ProdutoResource {

    private ProdutoBO produtoBo = new ProdutoBO();

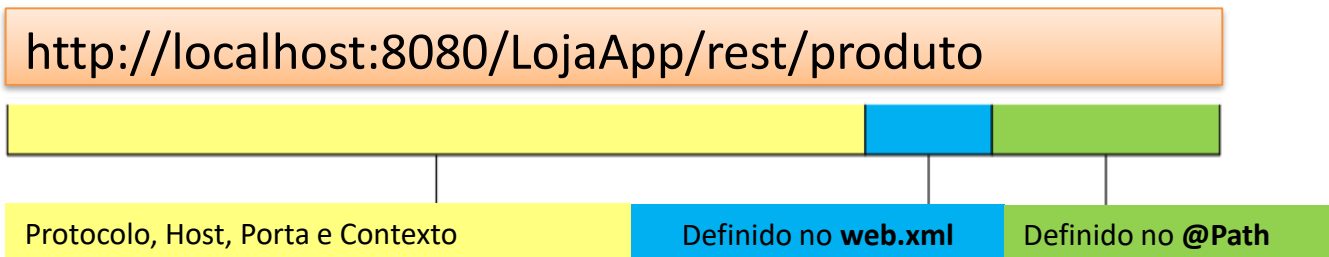
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<ProdutoTO> buscar(){
        return produtoBo.listar();
    }
}
```

← Tipo do retorno (JSON)

← Retorna a lista de produtos para ser convertido em um **JSON array**.



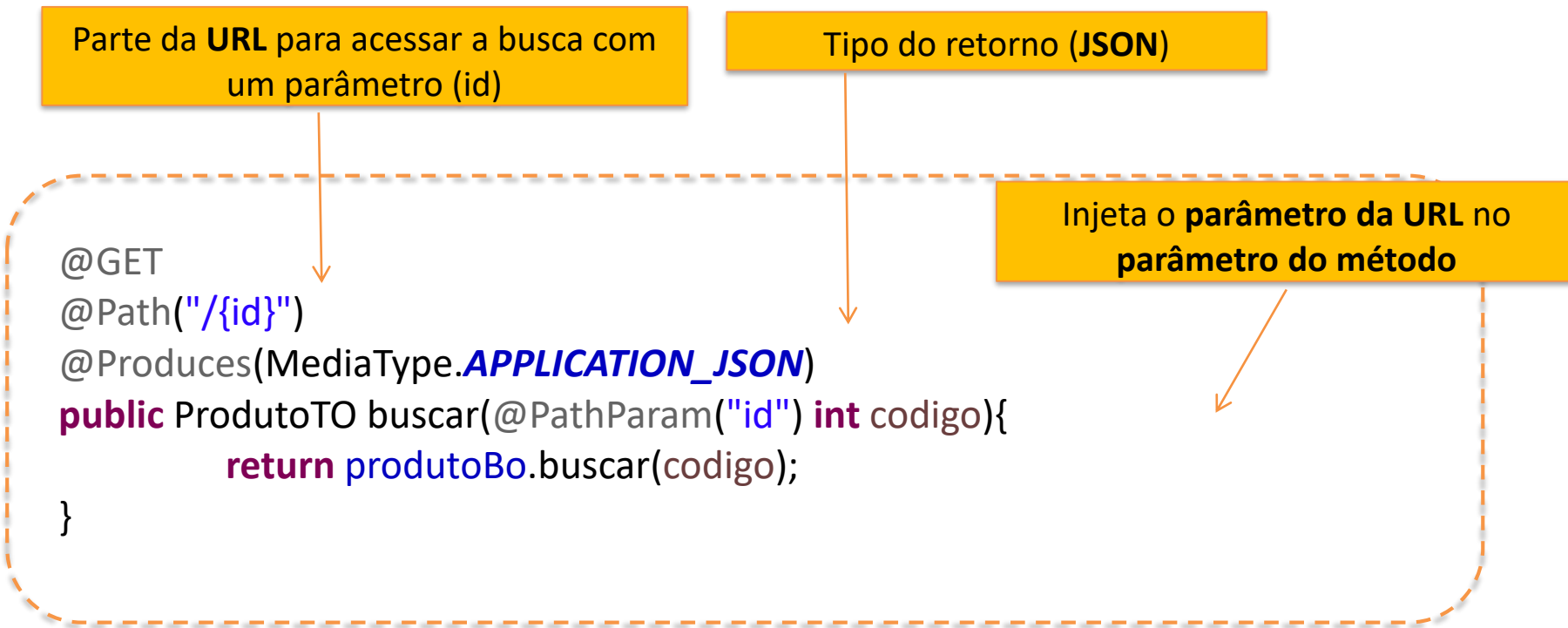
- O **método HTTP padrão** para acessar um endereço através do browser é o **GET**;
- Execute o Servidor e faça uma chamada ao serviço através de sua **URL** no navegador:



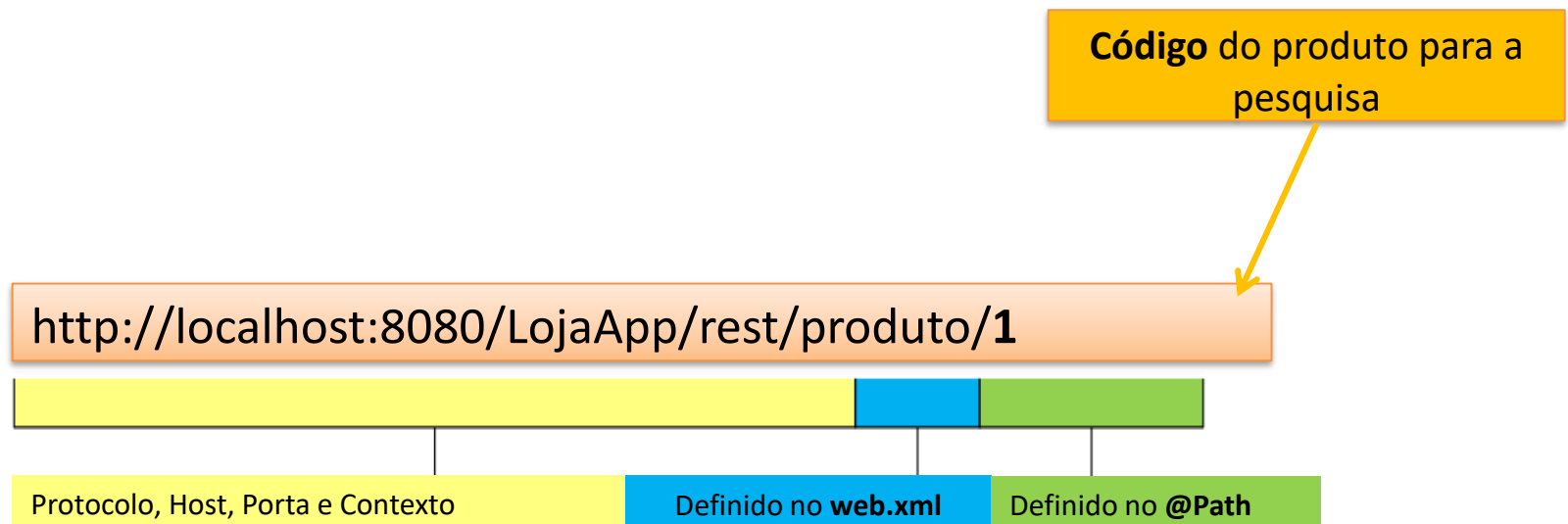
## Navegador:



- Adicione um serviço para **recuperar um produto** pelo seu código.



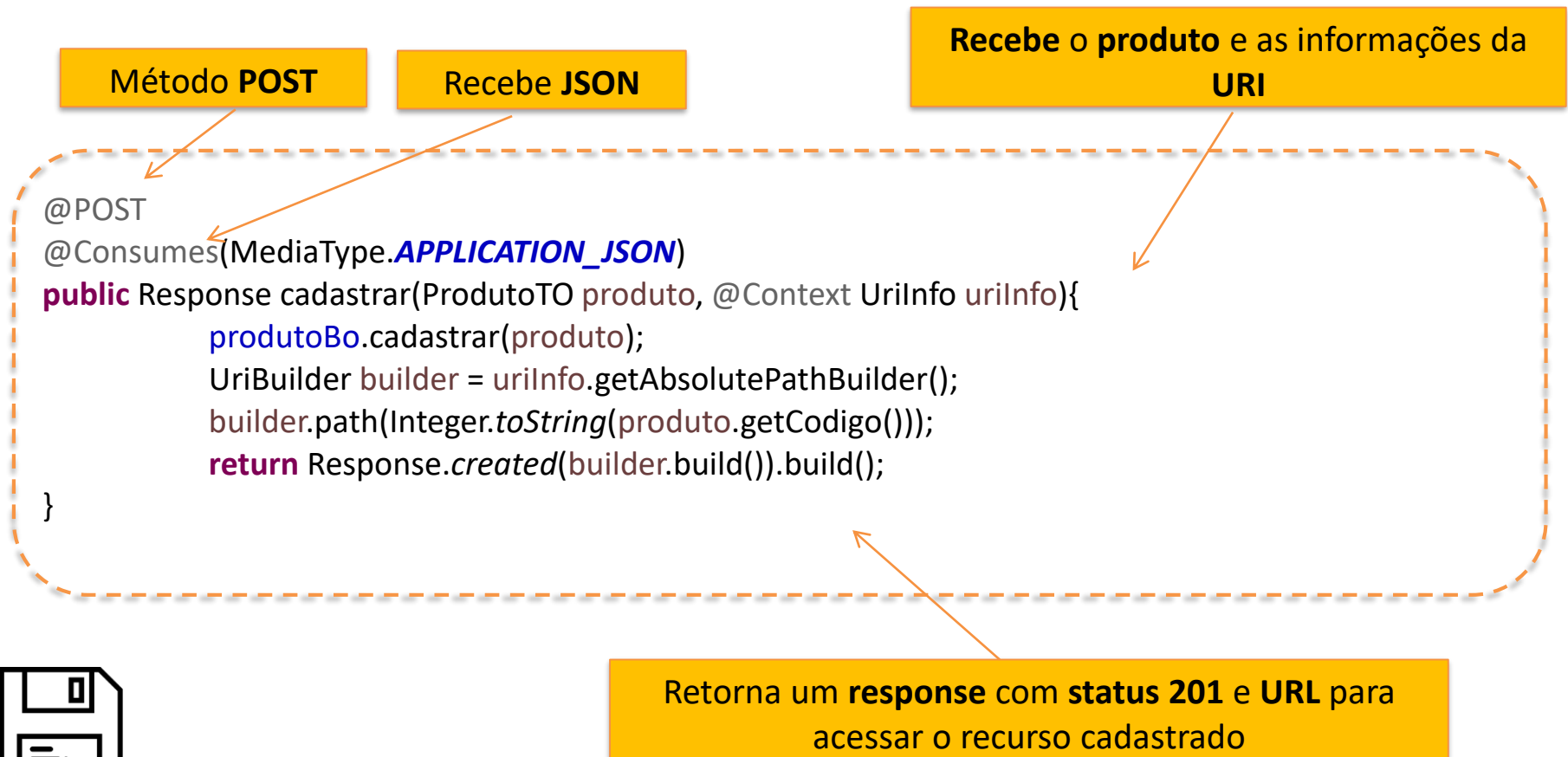
- Para realizar a **pesquisa por código**, basta adicionar o código do produto na **URL**;
- Dessa forma, se você informar o código, o web service **busca um produto específico**, caso não informe, é retornado **todos os produtos**;





# POST - CADASTRO

- Crie um método para **cadastrar** um Produto. O método recebe um objeto **ProdutoTO** e retorna um **Response**.



# POST – TESTE!

- Para enviar uma requisição POST, é preciso de uma ferramenta, como **Postman**.

The screenshot displays the Postman application interface. At the top, the URL bar shows 'http://localhost:8080/'. Below it, the 'POST' method is selected, and the URL is 'http://localhost:8080/LojaApp/rest/produto/'. The 'Body' tab is active, showing a JSON payload: 

```
{  "titulo": "Xiaomi",  "preco": 300,  "quantidade": 20}
```

. The 'Headers' tab is also visible, showing 'Content-Length → 0', 'Date → Fri, 10 Feb 2017 17:06:41 GMT', and 'Location → http://localhost:8080/LojaApp/rest/produto/4'. The status bar at the bottom indicates 'Status: 201 Created' and 'Time: 24 ms'.

**Método POST**

**URL para o Resource**

**Definição do conteúdo da requisição.**

**Conteúdo (Json)**

**Resposta recebida.**

**Link para acessar o recurso cadastrado.**



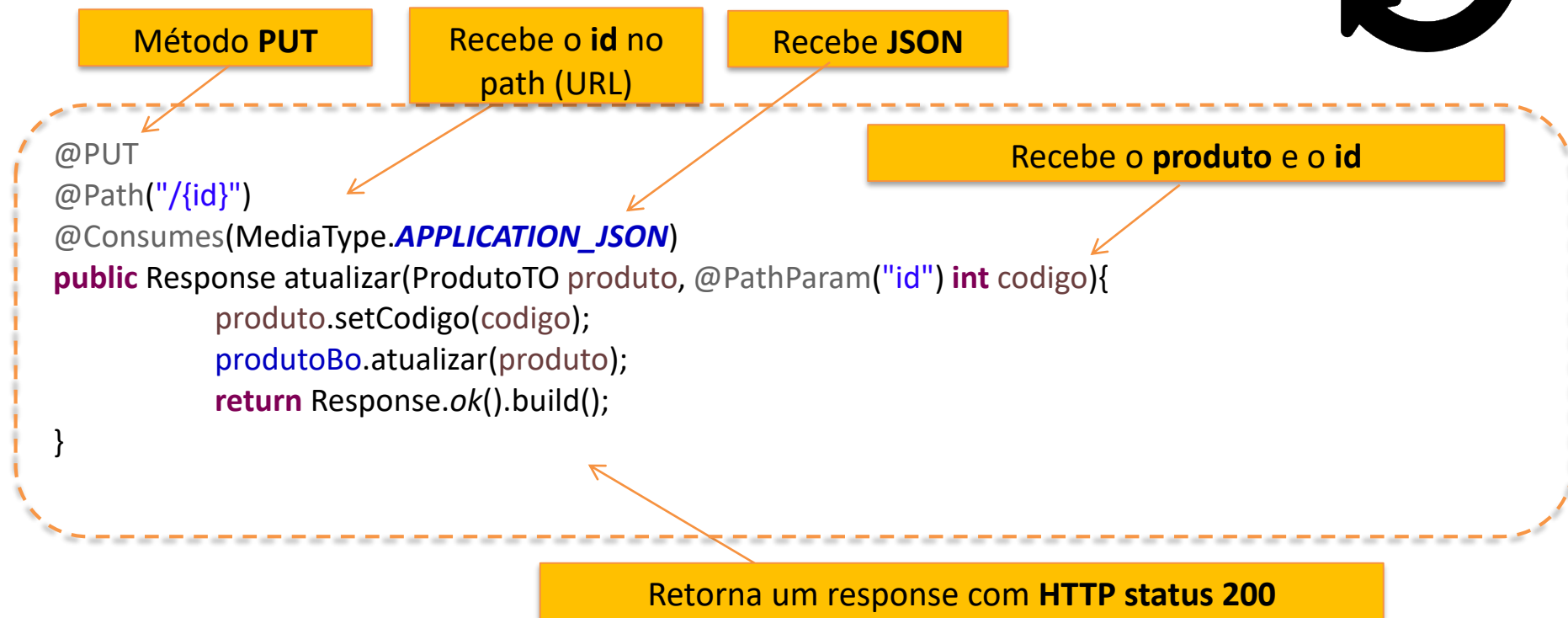
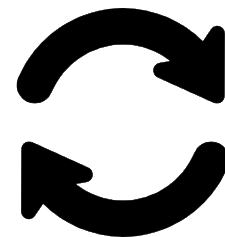
<https://www.getpostman.com/>





# PUT - ATUALIZAÇÃO

- O método para atualização recebe um objeto **ProdutoTO** e o **id** do produto que será atualizado.
- Retorna um **Response**.



- Para enviar uma requisição **PUT** vamos utilizar o **Postman**.

The screenshot displays the Postman interface for a PUT request. The URL bar shows `http://localhost:8080/LojaApp/rest/produto/1`. The method dropdown is set to **PUT**. The body is configured as **raw** with the content type **JSON (application/json)**. The JSON body contains the following data:

```
{
  "titulo": "Xiaomi",
  "preco": 300,
  "quantidade": 20
}
```

The response section at the bottom shows a status of **200 OK** and a time of **17 ms**.

**Método PUT**

**URL para o resource com o id**

**Definição do Conteúdo da requisição.**

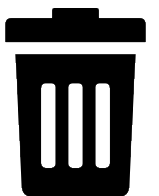
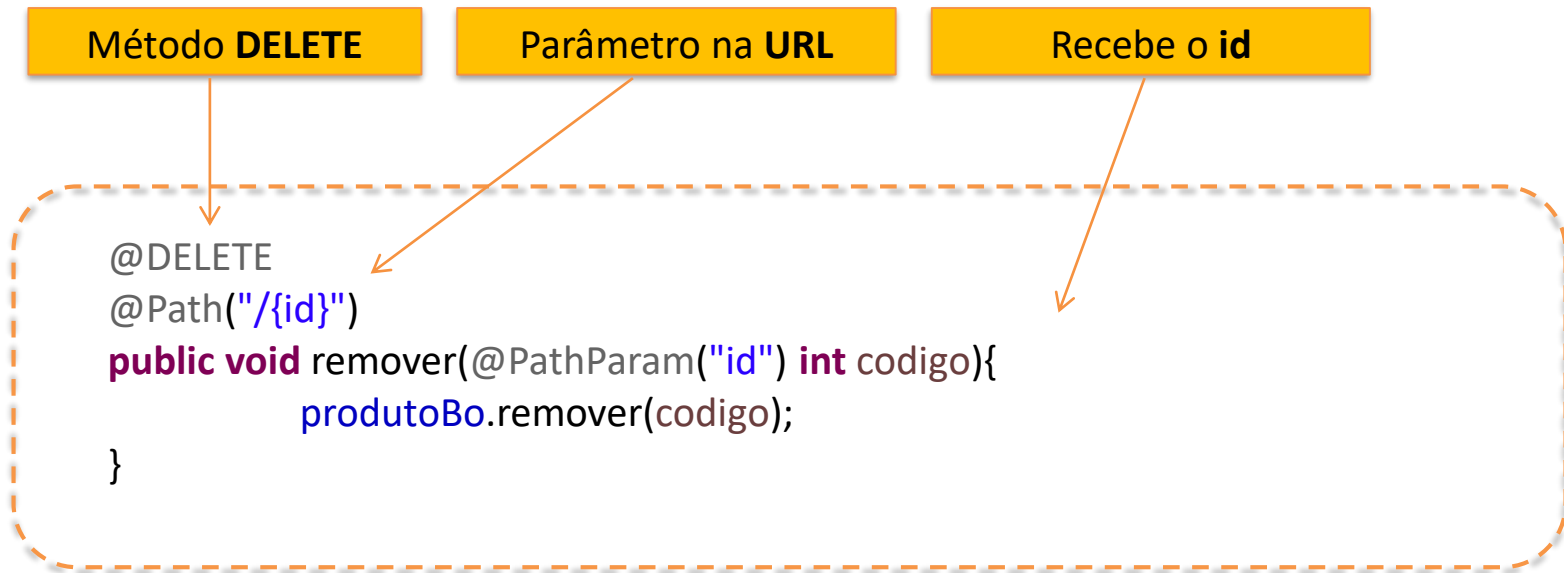
**Conteúdo (Json)**

**Resposta recebida.**



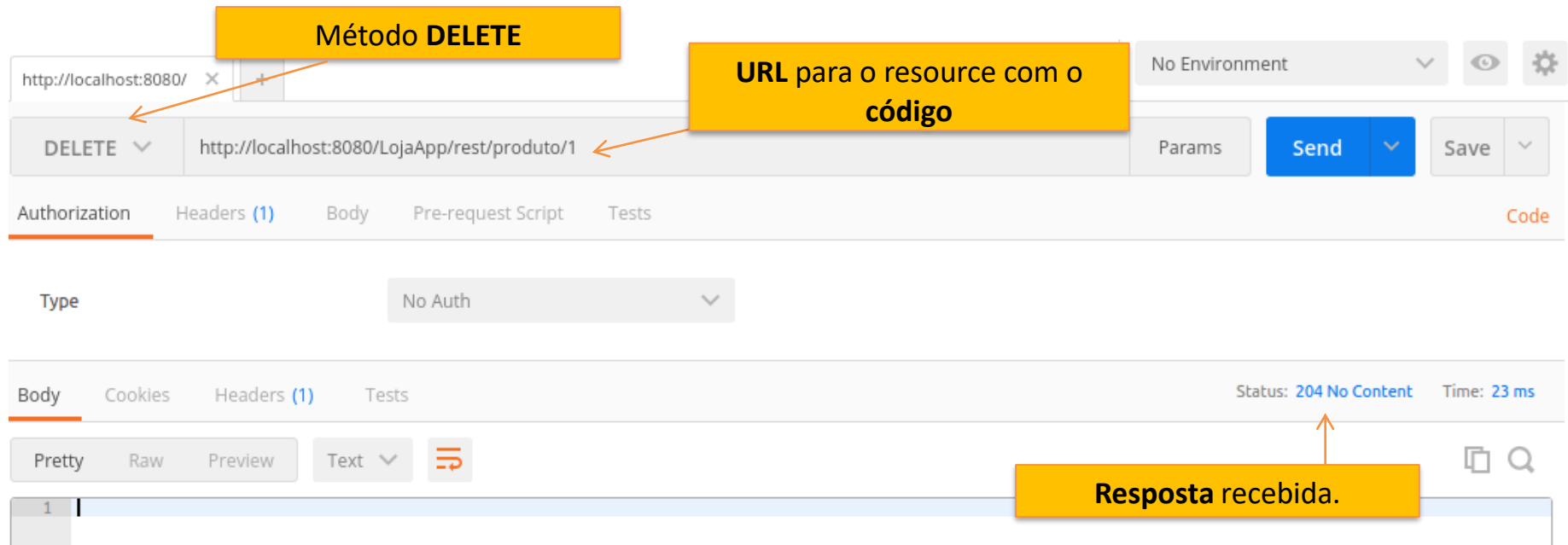
# DELETE - REMOÇÃO

- O método de remoção **recebe** um **código** e não retorna nada (código **HTTP 204** (no content));



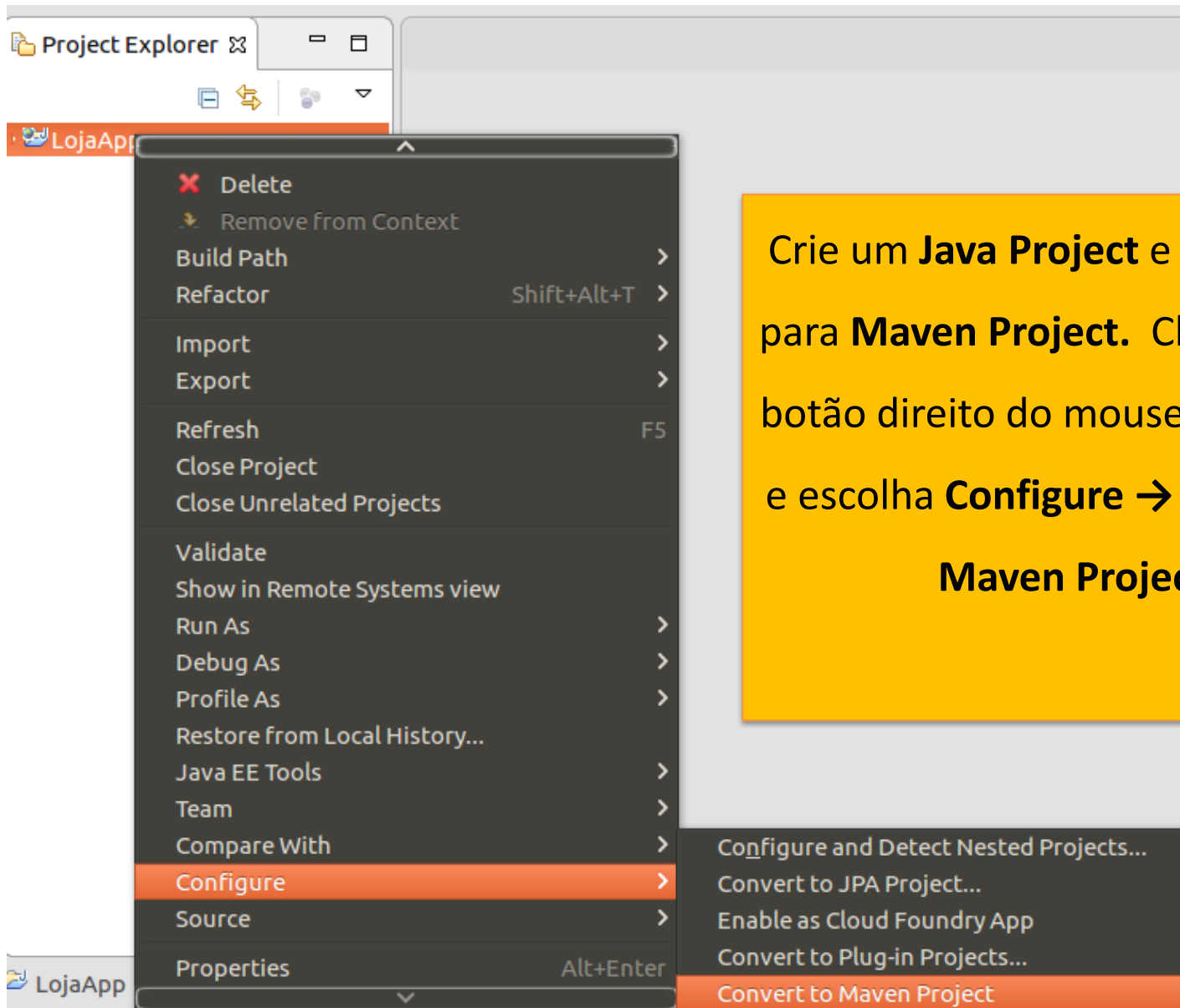
# | DELETE – TESTE!

- Utilize o **postman** para testar a função de **remoção**.





# CONSUMINDO UM WEB SERVICE RESTFUL EM JAVA



Crie um **Java Project** e converta-o para **Maven Project**. Clique com o botão direito do mouse no projeto e escolha **Configure → Convert to Maven Project**



- Configure o **pom.xml** para adicionar as dependências do projeto:

```
<dependencies>
  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
    <version>1.8</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>2.25.1</version>
  </dependency>
</dependencies>
```

Adicione as dependências após a tag `</build>`

- Crie uma classe para realizar a **listagem** de produtos:

```
Client client = Client.create();
```

URL para requisição

```
WebResource resource =  
    client.resource("http://localhost:8080/LojaApp/rest/produto");
```

Tipo de dado aceito

Requisição GET

```
ClientResponse response =  
    resource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
```

Verifica o status HTTP da resposta

```
if (response.getStatus() == 200){
```

Recupera o array de produtos

```
    ProdutoTO[] produtos = response.getEntity(ProdutoTO[].class);
```

```
    for (ProdutoTO produtoTO : produtos) {  
        System.out.println(produtoTO.getTitulo());
```

```
    }
```

```
}else{
```

```
    System.out.println("Erro - HTTP Status: " + response.getStatus());
```

```
}
```

- Exemplo de pesquisa de um **produto** pelo **código**:

```
Client client = Client.create();
```

```
WebResource resource =  
    client.resource("http://localhost:8080/LojaApp/rest/produto/1");
```

```
ClientResponse response =  
    resource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
```

```
if (response.getStatus() == 200){
```

```
    ProdutoTO produto = response.getEntity(ProdutoTO.class);  
    System.out.println(produto.getTitulo());
```

```
}else{
```

```
    System.out.println("Erro - HTTP Status: " + response.getStatus());
```

```
}
```

URL para requisição

Código do produto

Tipo de dado **aceito**

Requisição **GET**

Verifica o status **HTTP** da resposta

Recupera o **produto**

- Uma requisição **POST** para **cadastrar** um produto:

```
Client client = Client.create();
```

```
WebResource webResource =  
    client.resource("http://localhost:8080/LojaApp/rest/produto/");
```

```
ProdutoTO produto = new ProdutoTO(0, "Lenovo", 500, 10);
```

```
ClientResponse response =  
    webResource.type("application/json").post(ClientResponse.class, produto);
```

```
if (response.getStatus() == 201) {  
    System.out.println("Sucesso! " + response.getLocation());  
} else {  
    System.err.println("Erro: HTTP error code : " + response.getStatus());  
}
```

URL para requisição

Requisição **POST**

Envia o **produto**

Verifica o status **HTTP** da resposta

- Requisição **PUT** para **atualizar** um produto:

```
Client client = Client.create();
```

```
WebResource webResource =  
    client.resource("http://localhost:8080/LojaApp/rest/produto/1");
```

```
ProdutoTO produto = new ProdutoTO(0, "Xiaomi Mi", 300, 110);
```

```
ClientResponse response =  
    webResource.type("application/json").put(ClientResponse.class, produto);
```

```
if (response.getStatus() == 200) {  
    System.out.println("Sucesso! ");  
} else {  
    System.err.println("Erro: HTTP error code : " + response.getStatus());  
}
```

URL para requisição com o **código** do produto que será atualizado

Produto que será **atualizado**

Requisição **PUT**

Envia o **produto**

Verifica o status **HTTP** da resposta

- Para **remover** um produto, crie uma requisição **DELETE**:

```
Client client = Client.create();
```

```
WebResource webResource =  
    client.resource("http://localhost:8080/LojaApp/rest/produto/1");
```

URL para requisição com o **código** do produto que será removido

Requisição **DELETE**

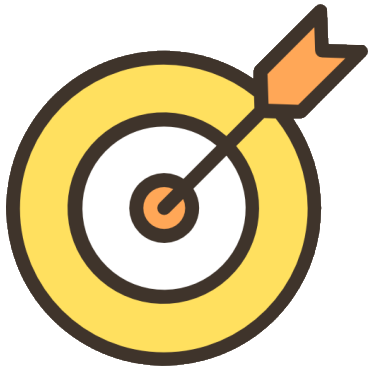
```
ClientResponse response = webResource.delete(ClientResponse.class);
```

```
if (response.getStatus() == 204) {  
    System.out.println("Sucesso! ");  
} else {  
    System.err.println("Erro: HTTP error code : " + response.getStatus());  
}
```

Verifica o status **HTTP** da resposta

## VOCÊ APRENDEU...

---



- Web services **Restful**, **recursos** e os **métodos HTTP** e seus **status** de retorno;
- **Implementar** um **web service Restful** com **JAX-RS** e **Json**;
- **Consumir** um **web service Restful** com **Java**;

**Copyright © 2018 – 2019**

**Prof. MSc. Rafael Matsuyama / Prof. Me. Thiago T. I. Yamamoto**

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*"Em nossas vidas, a mudança é inevitável. A perda é inevitável.  
A felicidade reside na nossa adaptabilidade em sobreviver a tudo de ruim" – Buda*