

Introdução ao AngularJS

Professor Anderson Henrique



Configurando o jersey no arquivo pom.xml

Precisamos configurar o Maven com Dependências do Jersey, como podemos saber quais dependências jersey devemos utilizar no Maven?

Acessando a documentação Jersey no seguinte endereço:

<https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html>

No item 2.3.2

[Links: Table of Contents](#) | [Single HTML](#)

Jersey 2.31 User Guide

Table of Contents

[Preface](#)

[1. Getting Started](#)

- [1.1. Creating a New Project from Maven Archetype](#)
- [1.2. Exploring the Newly Created Project](#)
- [1.3. Running the Project](#)
- [1.4. Creating a JavaEE Web Application](#)
- [1.5. Creating a Web Application that can be deployed on Heroku](#)
 - [1.5.1. Deploy it on Heroku](#)
- [1.6. Exploring Other Jersey Examples](#)

[2. Modules and dependencies](#)

- [2.1. Java SE Compatibility](#)
- [2.2. Introduction to Jersey dependencies](#)
- [2.3. Common Jersey Use Cases](#)
 - [2.3.1. Servlet based application on Glassfish](#)
 - [2.3.2. Servlet based server-side application](#)

```
1 <dependency>
2   <groupId>org.glassfish.jersey.containers</groupId>
3   <!-- if your container implements Servlet API older than 3.0, use "jersey-container-servlet-core" -->
4   <artifactId>jersey-container-servlet</artifactId>
5   <version>2.31</version>
6 </dependency>
```

Na seção de dependências vamos adicionar essa dependência, nosso arquivo pom.xml ficará assim

```
23 <dependency>
24   <groupId>org.glassfish.jersey.containers</groupId>
25   <artifactId>jersey-container-servlet</artifactId>
26   <version>2.0</version>
27 </dependency>
28 </dependencies>
29 </project>
```

Vamos limpar nosso arquivo pom.xml, deletando toda a seção <build> do nosso projeto.

Vamos alterar o <groupId> para com.crud.back e <version> para 1.0

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w
3     <modelVersion>4.0.0</modelVersion>
4
5     <groupId>com.crud.back</groupId>
6     <artifactId>CRUD-Back</artifactId>
7     <version>1.0</version>
8     <packaging>war</packaging>
9
10    <name>CRUD-Back</name>
```

Como configurar o jersey, como falar para o Java que estamos utilizando o jersey?

Vamos criar uma classe, um **startup** que inicia junto com o nosso projeto, essa classe será criada em um pacote que será informado no construtor dessa classe, crie um pacote na pasta Pacotes de Código-fonte chamado: **com.crud.back.controllers**

A classe será criada no pacote chamado: **com.crud.back** e o nome da classe é **MyApplication**

New Classe Java

1. Escolher Tipo de Arquivo

2. Nome e Localização

Nome da Classe: MyApplication

Projeto: CRUD-Back

Localização: Pacotes de Códigos-fonte

Pacote: com.crud.back

Arquivo Criado: uments\NetBeansProjects\CRUD-Back\src\main\java\com\crud\back\MyAp

< Voltar

Próximo >

Finalizar

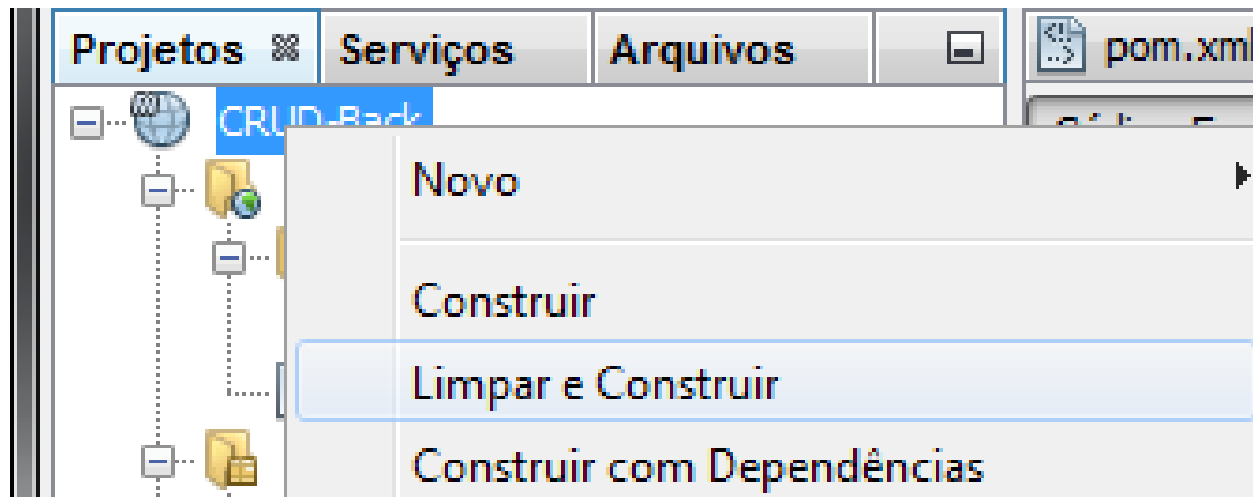
Cancelar

Ajuda



Limpar e Construir

Para que o Netbeans consiga fazer o download de todas as bibliotecas Jersey precisamos (Limpar e Construir nosso Projeto Java)



```

pom.xml [CRUD-Back]  MyApplication.java
Código-Fonte  Histórico
1  package com.crud.back;
2
3  import javax.ws.rs.ApplicationPath;
4  import org.glassfish.jersey.server.ResourceConfig;
5
6  @ApplicationPath("api")
7  public class MyApplication extends ResourceConfig {
8      public MyApplication() {
9          packages("com.crud.back.controllers");
10     }
11 }

```

Como sabemos que precisamos apenas dessa classe para configurar o jersey?

Acessando a documentação Jersey no seguinte endereço:

<https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html>

No item **4.2**

4. Application Deployment and Runtime Environments

4.1. Introduction

4.2. JAX-RS Application Model

Example 4.2. Reusing Jersey implementation in your custom application model

```
1 public class MyApplication extends ResourceConfig {  
2     public MyApplication() {  
3         packages("org.foo.rest;org.bar.rest");  
4     }  
5 }
```

Dentro do pacote **com.crud.back.controllers** vamos criar uma classe java chamada **HelloController**

Nesta classe teremos dois métodos, a nossa classe recebe uma anotação `@Path` que utilizamos para indicar o caminho da nossa classe

O primeiro método recebe a anotação `@GET` indicando que retornará um valor na requisição da URL, a anotação `@Produces` indica que o método produzirá um valor que é do tipo texto e vinculamos o valor do parâmetro na requisição da URL por meio da anotação `@QueryParam`

```
10  @Path("hello")
    public class HelloController {
12      @GET
13      @Produces(MediaType.TEXT_PLAIN)
    public String getMensagem(@QueryParam("usuario") String usuario){
15          return "Bem vindo: " + usuario;
16      }
```

O segundo método recebe a anotação `@GET` indicando que retornará um valor na requisição da URL, a anotação `@Produces` indica que o método produzirá um valor que é do tipo texto e a anotação `@Path` onde definimos um caminho para passagem do parâmetro.

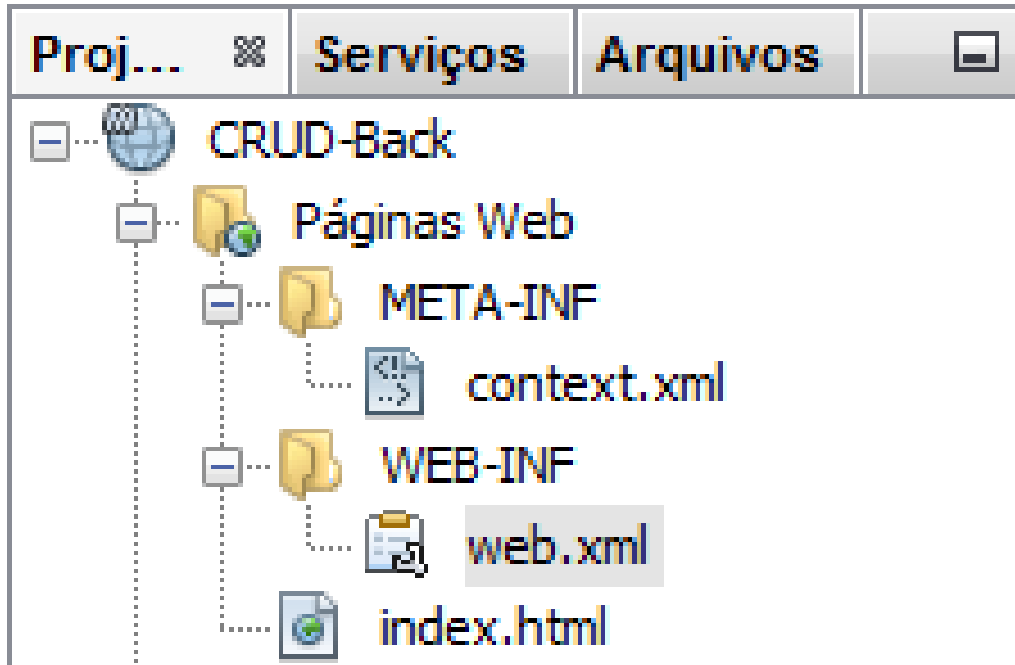
```
18     @GET
19     @Path("usuario/{id}")
20     @Produces(MediaType.TEXT_PLAIN)
21     public String getUsuario(@PathParam("id") long id) {
22         return "Recuperando usuário com ID: " + id;
23     }
24 }
```

Como vinculamos o parâmetro do método com o caminho será passado na URL? Por meio da anotação `@PathParam`

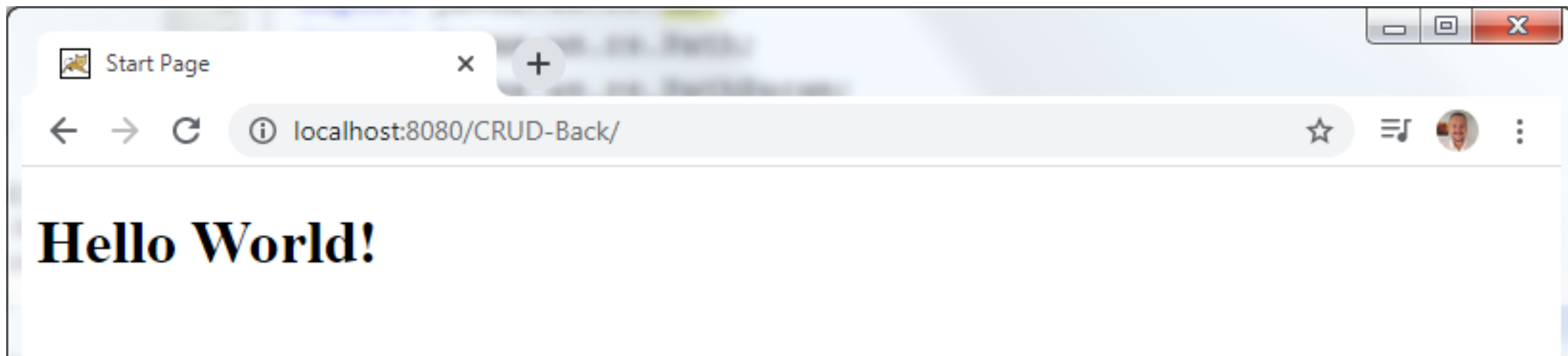
Ao Limpar e Construir o projeto, ocorrerá um erro na compilação informando que ainda não temos a pasta WEB-INF e o arquivo web.xml

Vamos criar no diretório Páginas Web uma pasta chamada **WEB-INF** e dentro desta pasta vamos criar um arquivo do tipo Descritor de Implantação Padrão (web.xml)

Nossa estrutura de pastas ficará assim:

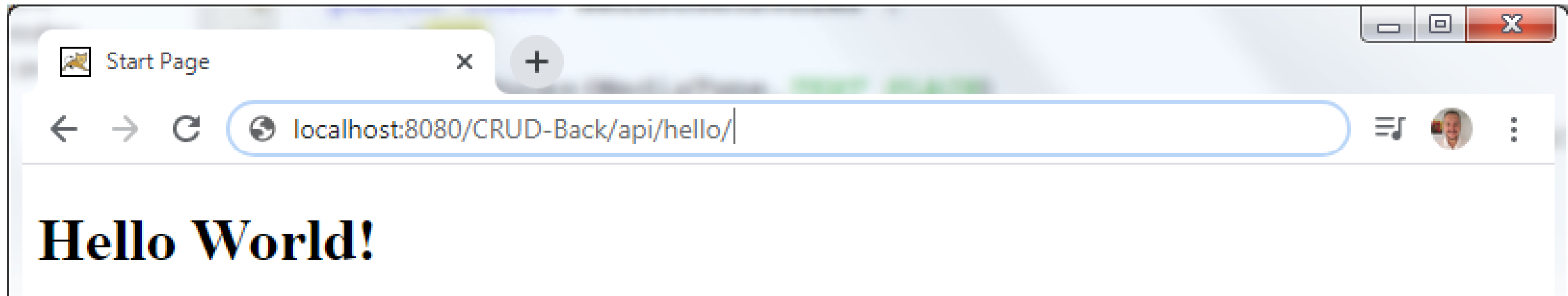


Vamos Limpar e Construir o projeto e executá-lo:

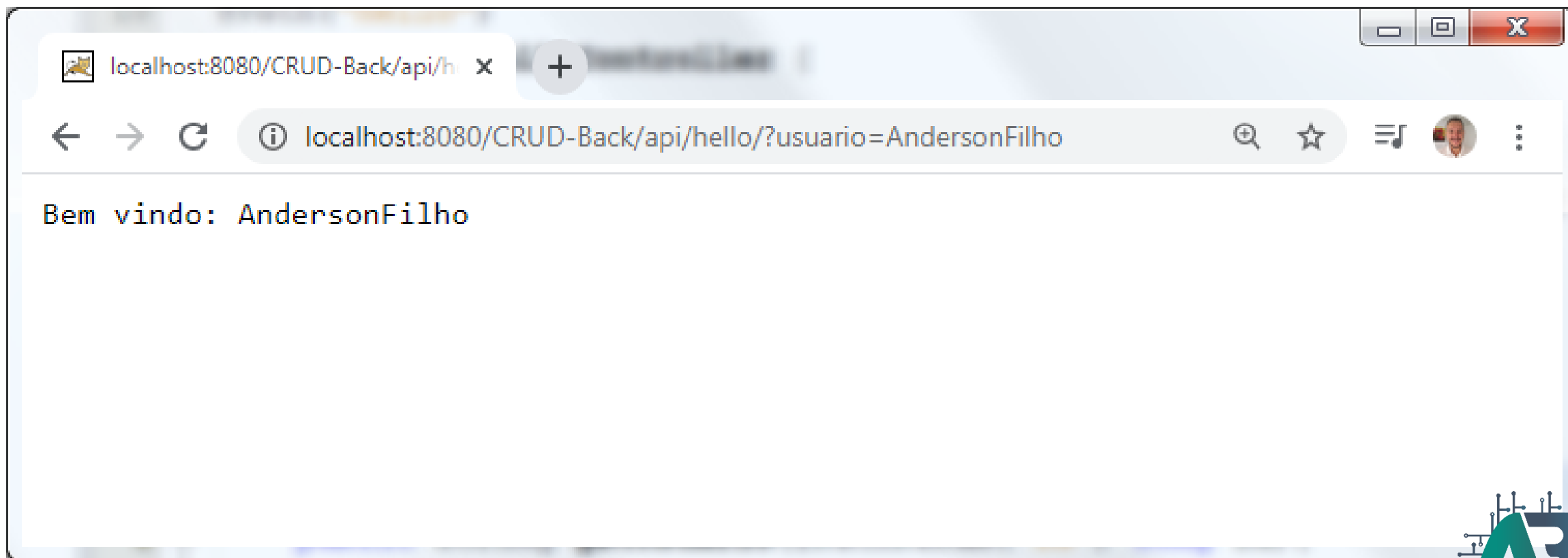


Na URL ele conseguiu executar nosso projeto no servidor Apache Tomcat **localhost:8080** e em seguida pegando o caminho do contexto da aplicação **CRUD-Back**

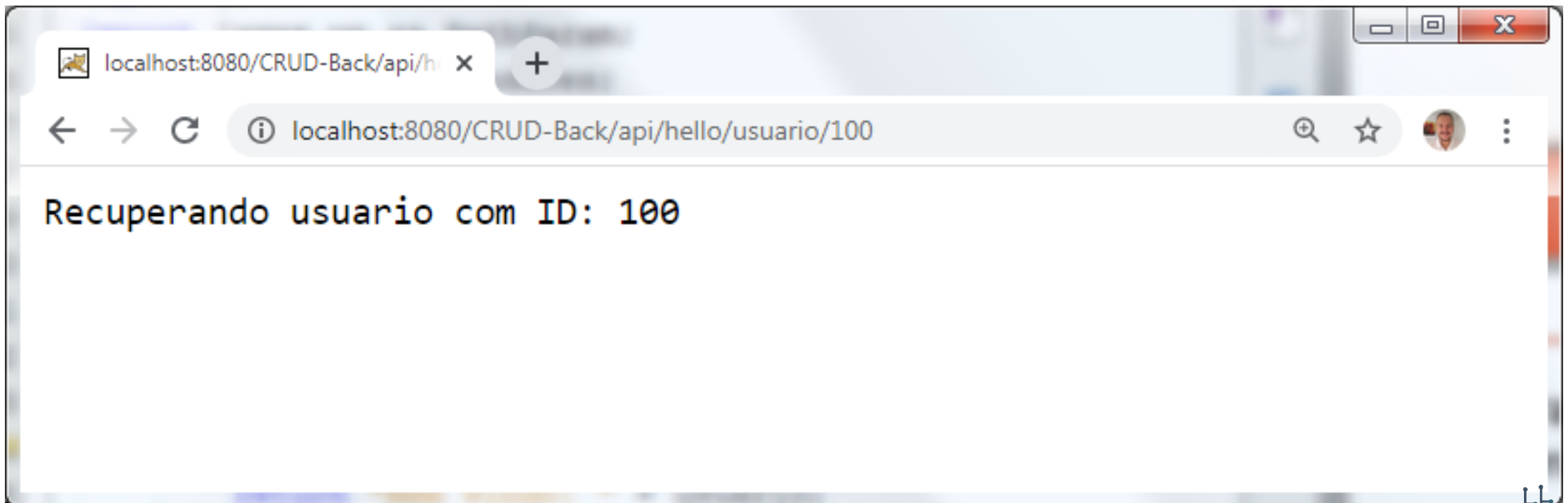
Para acessarmos a nossa aplicação precisamos seguir o caminho que foi definido na classe startup **MyApplication** e o caminho que foi definido na classe **HelloController**



Agora, quais dos dois métodos eu quero acessar? Acessando o **primeiro método** preciso passar um parâmetro na URL com o sinal ? Seguido do **nome do parâmetro** definido no método passando um valor com o sinal de =



Agora, quais dos dois métodos eu quero acessar? Acessando o **segundo método** preciso passar o caminho que foi definido no método e o valor do id do usuário



Dúvidas?

Professor Anderson Henrique



Para a próxima aula

01 – Aplicação Back-End em JAVA

02 – Criando nosso projeto CRUD (Create, Read, Update e Delete) em um Controller Restful API

Professor Anderson Henrique

