

Sistemas Distribuídos - Relatório

Universidade Federal do Ceará – Campus de Quixadá

Prática: Laboratório_API

Dupla: Mônica Maria Rodrigues

Maryana Moraes

Professor: Rafael Braga

1. Objetivo

O objetivo desta prática foi desenvolver uma API RESTful utilizando **Jakarta EE (JAX-RS)** e **EJB Stateless**, empacotar a aplicação em um arquivo **WAR**, e realizar o deploy em um servidor de aplicações **WildFly**.

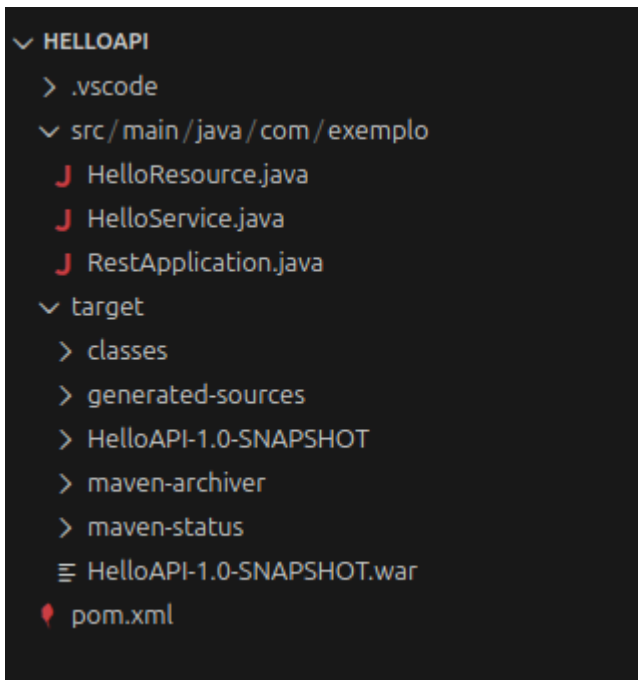
A API deveria responder a uma requisição HTTP retornando uma mensagem personalizada via método GET.

2. Ferramentas Utilizadas

- **Java 17**
- **Maven 3.3.2**
- **WildFly 30.0.0.Final**
- **Jakarta EE 10**
- **Terminal Linux (Ubuntu)**

3. Estrutura do Projeto

O projeto segue a arquitetura típica de aplicações Jakarta EE, organizado da seguinte forma:



4. Configuração do Maven (pom.xml)

O arquivo pom.xml foi configurado para:

1. Utilizar Jakarta EE 10;
2. Gerar corretamente um **arquivo WAR**;
3. Compilar com **Java 17**;
4. Empacotar o projeto como WAR usando o plugin apropriado.

```
pom.xml x J HelloResource.java J HelloService.java J RestApplication.java
pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.exemplo</groupId>
8   <artifactId>HelloAPI</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <packaging>war</packaging>
11
12
13  <dependencies>
14    <dependency>
15      <groupId>jakarta.platform</groupId>
16      <artifactId>jakarta.jakartaee-api</artifactId>
17      <version>10.0.0</version>
18      <scope>provided</scope>
19    </dependency>
20  </dependencies>
21
22  <build>
23    <plugins>
24
25      <!-- COMPILADOR - ESSENCIAL -->
26      <plugin>
27        <groupId>org.apache.maven.plugins</groupId>
28        <artifactId>maven-compiler-plugin</artifactId>
29        <version>3.11.0</version>
30        <configuration>
31          <source>17</source>
32          <target>17</target>
33        </configuration>
34      </plugin>
35
36      <!-- WAR -->
37      <plugin>
38        <groupId>org.apache.maven.plugins</groupId>
39        <artifactId>maven-war-plugin</artifactId>
40        <version>3.3.2</version>
41        <configuration>
42          <failOnMissingWebXml>false</failOnMissingWebXml>
43        </configuration>
44      </plugin>
45    </plugins>
46  </build>
47
48
49 </project>
```

- Após a configuração, o comando: **mvn clean package**
- gerou com sucesso o arquivo: **target/HelloAPI-1.0-SNAPSHOT.war**

5. Implementação da API

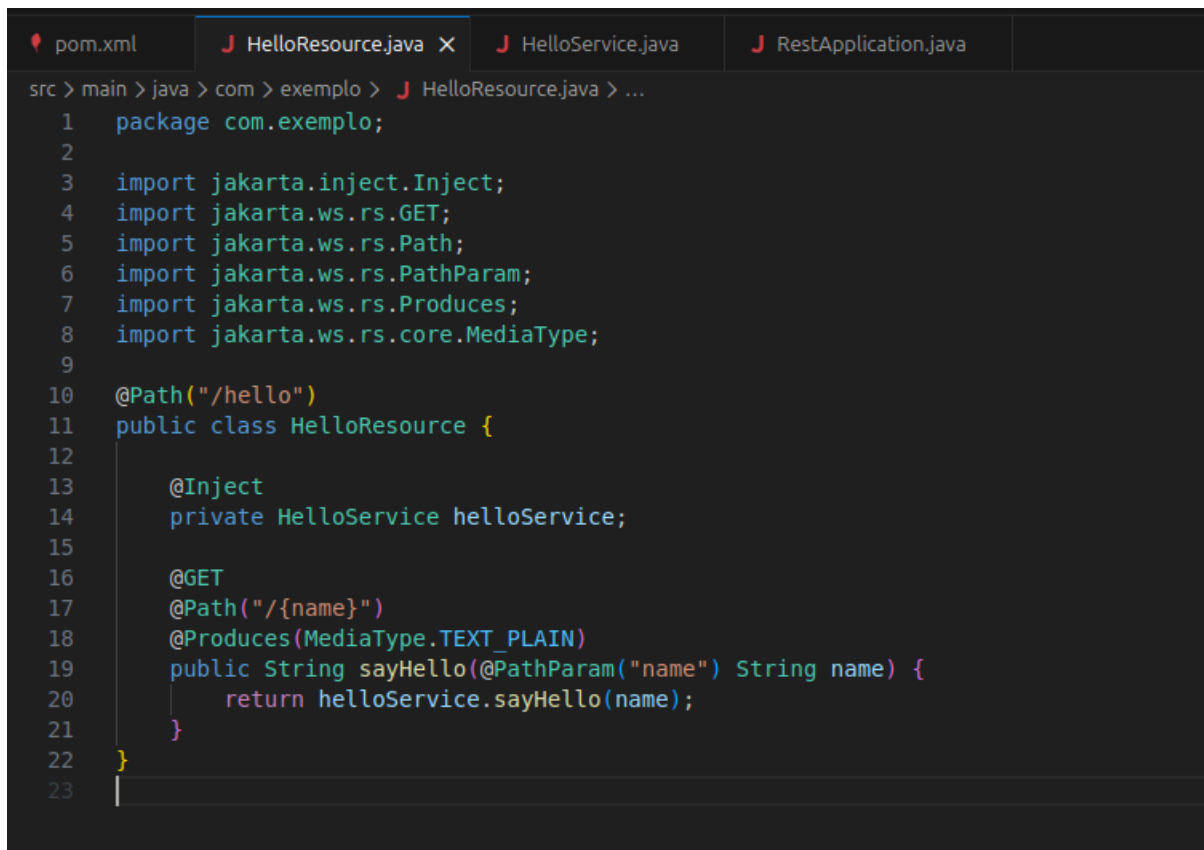
5.1 EJB Stateless – HelloService



```
pom.xml | HelloResource.java | HelloService.java X | RestApplication.java
src > main > java > com > exemplo > HelloService.java > ...
1 package com.exemplo;
2
3 import jakarta.ejb.Stateless;
4
5 @Stateless
6 public class HelloService {
7
8     public String sayHello(String name) {
9         return "Olá, " + name + "! Bem-vindo à API EJB.";
10    }
11 }
12
```

Esse componente fornece a lógica de negócio, retornando uma mensagem personalizada.

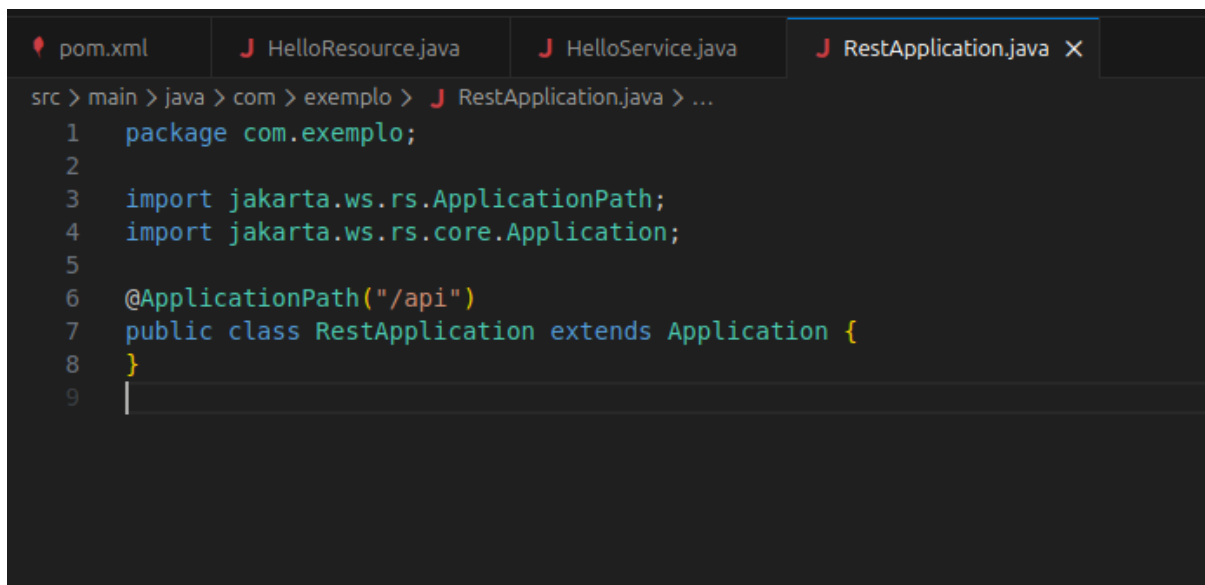
5.2 Recurso REST – HelloResource



```
pom.xml | HelloResource.java X | HelloService.java | RestApplication.java
src > main > java > com > exemplo > HelloResource.java > ...
1 package com.exemplo;
2
3 import jakarta.inject.Inject;
4 import jakarta.ws.rs.GET;
5 import jakarta.ws.rs.Path;
6 import jakarta.ws.rs.PathParam;
7 import jakarta.ws.rs.Produces;
8 import jakarta.ws.rs.core.MediaType;
9
10 @Path("/hello")
11 public class HelloResource {
12
13     @Inject
14     private HelloService helloService;
15
16     @GET
17     @Path("/{name}")
18     @Produces(MediaType.TEXT_PLAIN)
19     public String sayHello(@PathParam("name") String name) {
20         return helloService.sayHello(name);
21     }
22 }
23
```

Responsável por expor o endpoint REST via HTTP GET.

5.3 Configuração da API – RestApplication



```
src > main > java > com > exemplo > RestApplication.java > ...
1  package com.exemplo;
2
3  import jakarta.ws.rs.ApplicationPath;
4  import jakarta.ws.rs.core.Application;
5
6  @ApplicationPath("/api")
7  public class RestApplication extends Application {
8  }
9  |
```

Esta classe define o caminho base /api para todos os recursos REST.

6. Instalação e Execução no WildFly

1. O WildFly 30.0.0.Final foi baixado e extraído.
2. A pasta foi organizada como:

~/wildfly

3. O servidor foi iniciado:

./standalone.sh

4. Após a inicialização, o log apresentou:
WFLYSRV0025: WildFly Full 30.0.0.Final started

5. O arquivo WAR foi enviado para o WildFly:
cp target/HelloAPI-1.0-SNAPSHOT.war ~/wildfly/standalone/deployments/

6. O servidor criou automaticamente o arquivo:

HelloAPI-1.0-SNAPSHOT.war.deployed

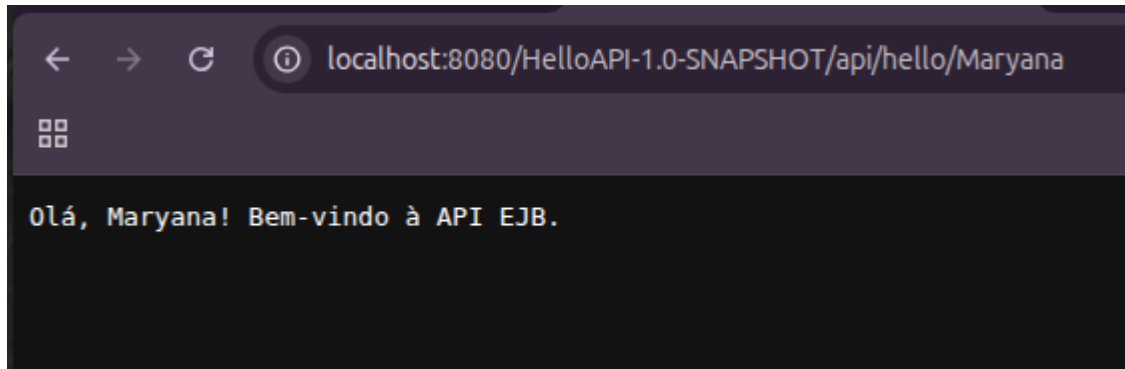
indicando que o deploy foi bem-sucedido.

7. Teste da API

Com o servidor rodando, a API foi acessada pelo navegador usando:

<http://localhost:8080/HelloAPI-1.0-SNAPSHOT/api/hello/Maryana>

A resposta obtida foi:



Confirmando que o EJB foi injetado corretamente e o endpoint REST está funcionando.

8. Conclusão

A prática foi concluída com sucesso.

Foi possível desenvolver uma aplicação completa utilizando:

- EJB Stateless
- JAX-RS
- Jakarta EE 10
- WildFly
- Maven com empacotamento WAR

A aplicação foi implantada e acessada corretamente via HTTP, demonstrando domínio dos conceitos de desenvolvimento e deploy de aplicações corporativas Java.