# Sistemas Distribuídos - Relatório

**Universidade Federal do Ceará – Campus de Quixadá**

**Prática:** Laboratório_API

**Dupla:** Mônica Maria Rodrigues

Maryana Moraes

**Professor:** Rafael Braga

## 1. Objetivo

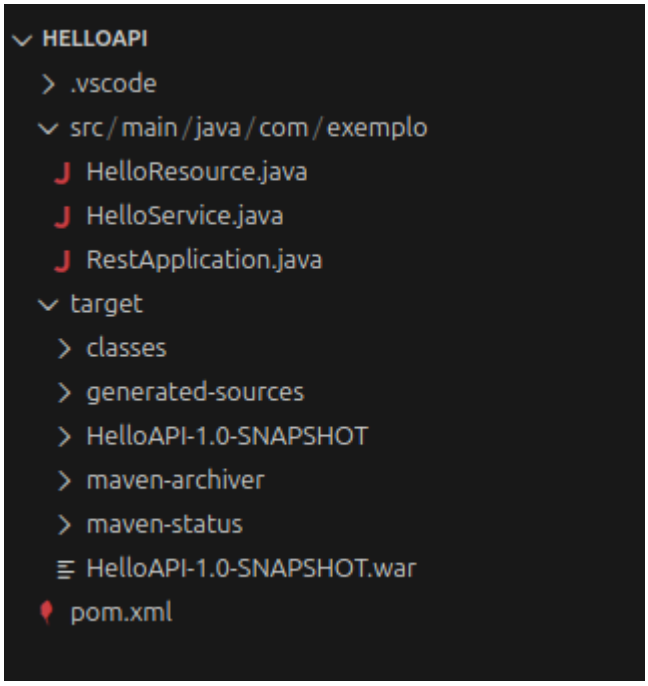
O objetivo desta prática foi desenvolver uma API RESTful utilizando **Jakarta EE (JAX-RS)** e **EJB Stateless**, empacotar a aplicação em um arquivo **WAR**, e realizar o deploy em um servidor de aplicações **WildFly**.
A API deveria responder a uma requisição HTTP retornando uma mensagem personalizada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

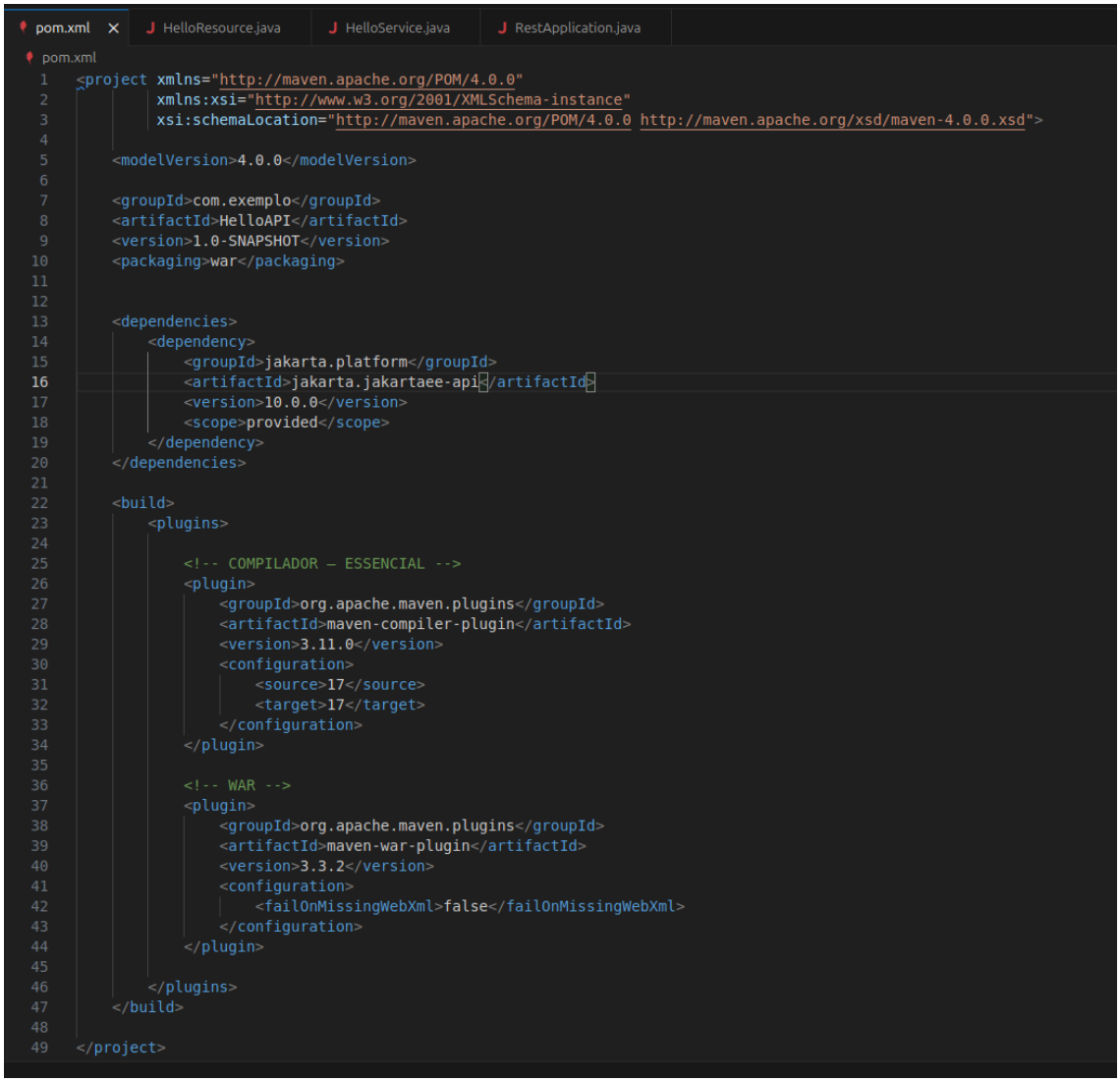
O projeto segue a arquitetura típica de aplicações Jakarta EE, organizado da seguinte forma:




## 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.




- 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 | RestApplication.java

src > main > java > com > exemplo > HelloService.java > ...

```java
package com.exemplo;

import jakarta.ejb.Stateless;

@Stateless
public class HelloService {

    public String sayHello(String name) {
        return "Olá, " + name + "! Bem-vindo à API EJB.";
    }
}
```

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

## 5.2 Recurso REST – HelloResource

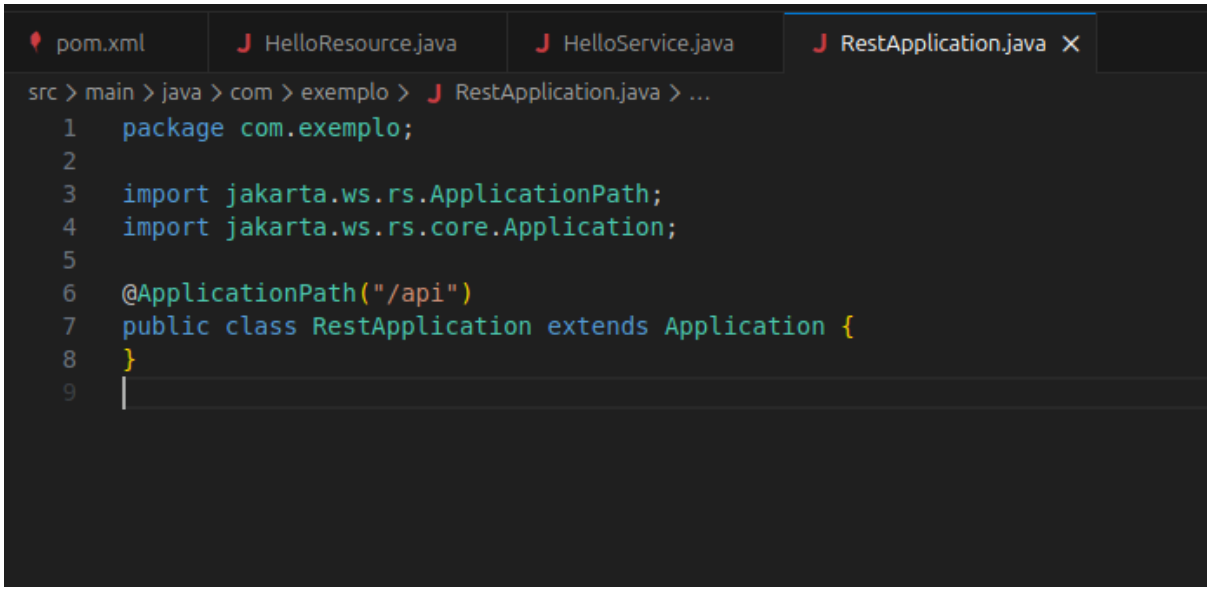
pom.xml | HelloResource.java | HelloService.java | RestApplication.java

src > main > java > com > exemplo > HelloResource.java > ...

```java
package com.exemplo;

import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.PathParam;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/hello")
public class HelloResource {

    @Inject
    private HelloService helloService;

    @GET
    @Path("/{name}")
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello(@PathParam("name") String name) {
        return helloService.sayHello(name);
    }
}
```

Responsável por expor o endpoint REST via HTTP GET.

### 5.3 Configuração da API – RestApplication




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**

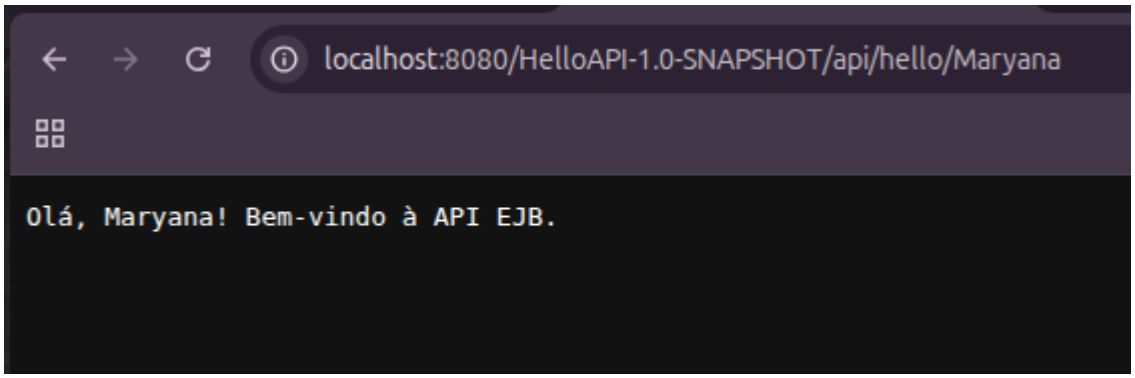
   indicando que o deploy foi bem-sucedido.

## 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.