



UNIVERSIDADE FEDERAL DO CEARÁ

11, Julho 2025

Sistemas Distribuídos - 01A - 2025.1 Lab. API RESTful

Monalisa Silva Bezerra - 535614

Professor: Dr. Antonio Rafael Braga

1. Objetivo

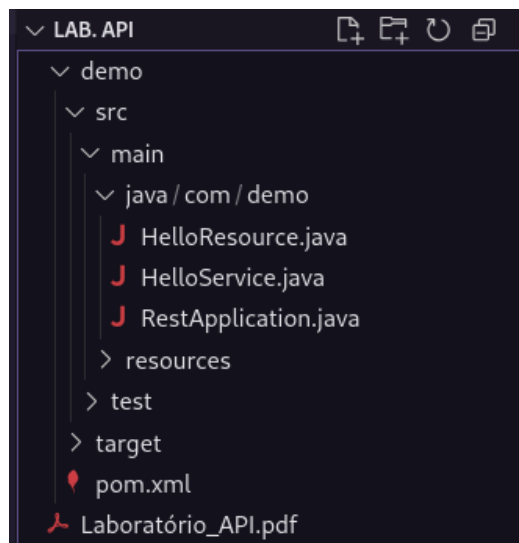
Construir uma API RESTful simples em Java, usando:

- **EJB Stateless** para lógica de negócio;
- **JAX-RS (Jakarta EE)** para exposição de endpoints HTTP;
- **WildFly** como servidor de aplicações.

2. Etapas Realizadas

2.1. Configuração do projeto Maven

- I. Foi criado o diretório do projeto e inicializou um artefato do tipo WAR.



- II. No `pom.xml`, incluiu-se a dependência Jakarta EE 10 no escopo *provided*, garantindo acesso a todas as APIs necessárias sem empacotá-las no WAR.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.demo</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <!-- Define a versão Java para compilação -->
  <properties>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!-- Jakarta EE completo -->
    <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>10.0.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>

      <!-- Gera o WAR sem exigir web.xml -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.2</version>
        <configuration>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>

      <!-- Compilador Java -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>


    </plugins>
  </build>

</project>
```

- III. Adicionou-se o plugin `maven-war-plugin` para gerar o WAR mesmo sem `web.xml`.

2.2. Implementação do EJB Stateless

- I. Arquivo: `HelloService.java`.
- II. Local: `src/main/java/com/demo/`.
- III. Conteúdo:



```
package com.demo;

import jakarta.ejb.Stateless;

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

2.3. Criação do recurso REST (JAX-RS)

- I. Arquivo: `HelloResource.java`.
- II. Local: `src/main/java/com/demo/`.
- III. Conteúdo:

```

package com.demo;

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);
    }
}

```

2.4. Configuração da aplicação JAX-RS

- I. Arquivo: `RestApplication.java`.
- II. Local: `src/main/java/com/demo/`.
- III. Conteúdo:

```

package com.demo;

import jakarta.ws.rs.ApplicationPath;
import jakarta.ws.rs.core.Application;

@ApplicationPath("/api")
public class RestApplication extends Application {
    // Não precisa implementar nada aqui
}

```

3. Build e Deploy

- I. O projeto foi compilado e gerado o WAR: `mvn clean package`
- II. Foi feito o deploy no WildFly: `cp target/HelloAPI.war /standalone/deployments/`

4. Testes e Resultados

- I. URL: `http://localhost:8080/HelloAPI/api/hello/Monalisa`
- II. Gerou: Olá, Monalisa! Bem-vindo à API EJB.
- III. A API respondeu corretamente às solicitações:
 - O EJB foi injetado e executado;
 - O JAX-RS mapeou as rotas e produziu texto simples;
 - O WildFly hospedou o serviço sem necessidade de configurações adicionais.

5. Conclusão

O laboratório foi finalizado demonstrando:

- Como configurar um projeto Jakarta EE com Maven;
- A criação e injeção de um EJB Stateless;
- A exposição de métodos de negócio via JAX-RS;
- O ciclo completo de build, deploy e teste em WildFly.

Todos os objetivos foram atendidos, e a API está pronta para evoluções (novos recursos, formatos JSON, autenticação, etc.).