

# Desenvolvimento de APIs REST

## 02 - Spring Boot

- Maven
- Spring Boot
- Controllers
- Postman

Apache Maven, ou simplesmente Maven, é uma ferramenta de automação de compilação, sendo utilizada primariamente em projetos Java, mas também usados em outras linguagens, como C#, Ruby e Scala.

## Maven



# Maven - POM.XML



## Introdução Maven/Pom

Utiliza um arquivo XML para descrever o projeto a ser construído. Este arquivo se chama POM (Project Object Model), e nele são descritas informações, tais como dependências, componentes, ordem de compilação, pastas e plugins necessários para a construção do projeto.

```
<project>
  <groupId>br.org.serratec</groupId>
  <artifactId>e-commerce</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>e-commerce project</name>
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        //...
      </plugin>
    </plugins>
  </build>
</project>
```

# Maven - POM.XML



## Identificadores do projeto

- **groupId** – um nome único para identificar a empresa ou grupo como único, sendo geralmente utilizado a url do site da empresa, assim como os pacotes em java
- **artifactId** – identificador único do projeto
- **version** – versão do projeto
- **packaging** – método de empacotamento da aplicação (WAR/JAR/ZIP)

```
<project>
  <groupId>br.org.serratec</groupId>
  <artifactId>e-commerce</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>e-commerce project</name>
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        //...
      </plugin>
    </plugins>
  </build>
</project>
```

# Maven - POM.XML



## Dependências

São bibliotecas externas que a aplicação necessita, sendo baixadas de um repositório na Internet e armazenada em um repositório local (na pasta .m2 dentro da pasta do usuário no sistema operacional).

Estas dependências são referenciadas pelos seus indicadores de projeto dentro do bloco dependencies no pom.xml

```
<project>
  <groupId>br.org.serratec</groupId>
  <artifactId>e-commerce</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>e-commerce project</name>
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        //...
      </plugin>
    </plugins>
  </build>
</project>
```

# Maven - POM.XML



## Propriedades

São como variáveis que definem os valores que podem ser utilizados em vários pontos do arquivo pom.xml. Algumas variáveis já são definidas pelo próprio Maven (como `project.build.directory`). O desenvolvedor também pode definir as suas próprias variáveis. Exemplo: definir as versões das dependências que serão reutilizadas em vários pontos.

```
<project>
...
<properties>
  <spring.version>5.3.16</spring.version>
  <project.build.folder>${project.build.directory}/tmp/</project.build.folder>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>
</dependencies>
<plugin>
  //...
  <outputDirectory>${project.resources.build.folder}</outputDirectory>
  //...
</plugin>
</project>
```

# Maven - POM.XML



## Build

Permite configurar diversas informações sobre a construção do projeto, incluindo utilização de plugins específicos, como o do Spring Boot

*(basedir é uma propriedade do maven que indica a pasta onde o projeto se encontra)*

```
<project>
  //...
  <build>
    <defaultGoal>install</defaultGoal>
    <directory>${basedir}/target</directory>
    <finalName>${artifactId}-${version}</finalName>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      //...
    </plugins>
    //...
  </build>
</project>
```

# Maven - Repositórios



## Repositórios

Por padrão, o maven realiza sempre o download das bibliotecas do seu repositório central (<https://repo1.maven.org/maven2/>), mas é possível acrescentar outros repositórios no pom.xml

```
<project>
  <groupId>br.org.serratec</groupId>
  <artifactId>e-commerce</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>e-commerce project</name>
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <repositories>
    <repository>
      <id>JBoss repository</id>
      <url>http://repository.jboss.org/nexus/content/groups/public</url>
    </repository>
  </repositories>
  ...
  ...
</project>
```



# Maven - Repositórios



Indexador de repositórios maven. Auxilia na pesquisa de dependências para inclusão no pom.xml

<https://mvnrepository.com/>

The screenshot shows the Maven Repository website. On the left, there's a sidebar with 'Indexed Artifacts (17.6M)' and 'Popular Categories'. The main content area is titled 'What's New in Maven' and lists several WSO2 artifacts, including 'WSO2 Streaming Integrator Interceptors' and 'WSO2 Carbon Data Provider Feature'. A red box highlights the 'Indexed Repositories (1288)' section, which lists various repositories like Central, Sonatype, Spring Plugins, etc. A red arrow points from this section to the text below.

Repositórios indexados (“Central” é o principal repositório Maven)

The screenshot shows the page for 'Spring Boot Starter Web >> 2.3.2.RELEASE'. It includes a description, license (Apache 2.0), organization (Pivotal Software, Inc.), homepage, date, files, and repositories. A table at the bottom shows 'Used By' with 5,668 artifacts. A red box highlights the 'Maven' tab, which contains a code block for a Maven dependency. A red arrow points from this code block to the text below.

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.3.2.RELEASE</version>
</dependency>
```

Bloco de dependência para “copiar” e “colar” no pom.xml

# Maven - Repositório Local



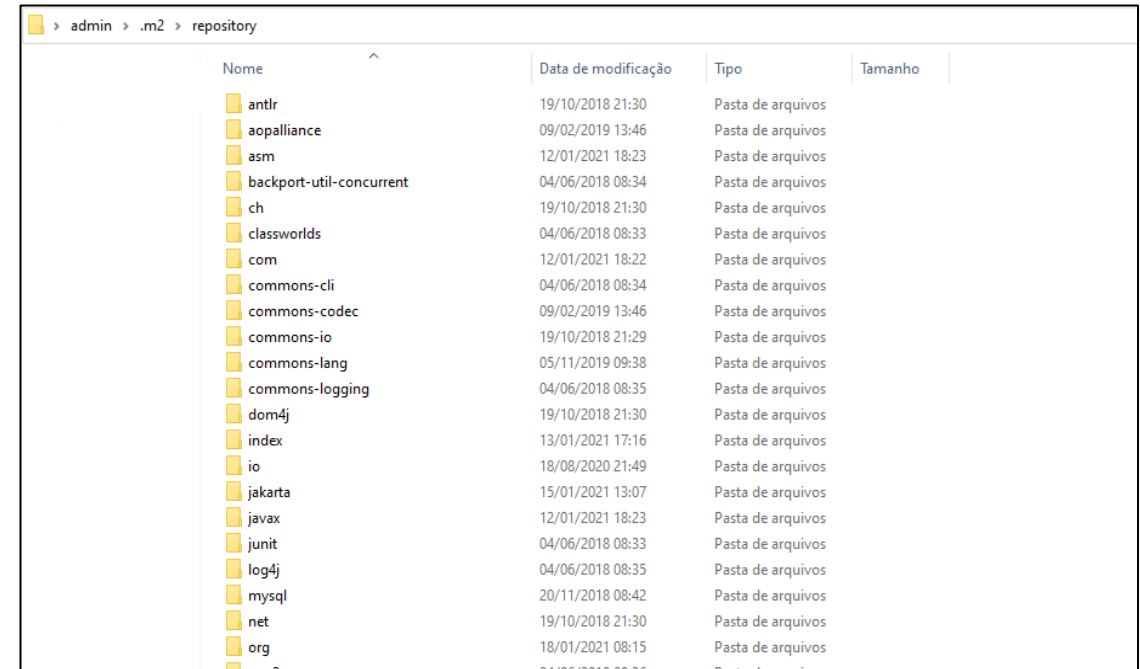
**Repositório local:** <home do usuário>/m2

**<home do usuário>**

- Windows: C:\Users\<usuário>
- Linux: /home/<usuário>
- Mac: /Users/<usuário>

Bibliotecas são baixadas uma vez e compartilhada entre todos os projetos maven na máquina

Estrutura de diretórios segue os valores do pom: groupId, artifactId e número de versão



Nome	Data de modificação	Tipo	Tamanho
antlr	19/10/2018 21:30	Pasta de arquivos	
aopalliance	09/02/2019 13:46	Pasta de arquivos	
asm	12/01/2021 18:23	Pasta de arquivos	
backport-util-concurrent	04/06/2018 08:34	Pasta de arquivos	
ch	19/10/2018 21:30	Pasta de arquivos	
classworlds	04/06/2018 08:33	Pasta de arquivos	
com	12/01/2021 18:22	Pasta de arquivos	
commons-cli	04/06/2018 08:34	Pasta de arquivos	
commons-codec	09/02/2019 13:46	Pasta de arquivos	
commons-io	19/10/2018 21:29	Pasta de arquivos	
commons-lang	05/11/2019 09:38	Pasta de arquivos	
commons-logging	04/06/2018 08:35	Pasta de arquivos	
dom4j	19/10/2018 21:30	Pasta de arquivos	
index	13/01/2021 17:16	Pasta de arquivos	
io	18/08/2020 21:49	Pasta de arquivos	
jakarta	15/01/2021 13:07	Pasta de arquivos	
javax	12/01/2021 18:23	Pasta de arquivos	
junit	04/06/2018 08:33	Pasta de arquivos	
log4j	04/06/2018 08:35	Pasta de arquivos	
mysql	20/11/2018 08:42	Pasta de arquivos	
net	19/10/2018 21:30	Pasta de arquivos	
org	18/01/2021 08:15	Pasta de arquivos	



```
> spring-boot-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot\2.4.2
> spring-boot-autoconfigure-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-autoconfigure\2.4.2
> spring-boot-starter-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter\2.4.2
> spring-boot-starter-json-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter-json\2.4.2
> spring-boot-starter-logging-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter-logging\2.4.2
> spring-boot-starter-test-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter-test\2.4.2
> spring-boot-starter-tomcat-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter-tomcat\2.4.2
> spring-boot-starter-web-2.4.2.jar - C:\Users\admin\.m2\repository\org\springframework\boot\spring-boot-starter-web\2.4.2
```

# Maven - Ciclo de Vida



Durante a construção de um projeto maven, um determinado ciclo de vida é seguido, pois é composto por diversos “goals” (etapas, objetivos, metas...):

- validate - verifica se o projeto está correto
- compile - compila o projeto em “binários”
- test - executa os testes unitários \*
- package - empacota o código compilado no tipo de empacotamento definido (JAR/WAR/ZIP)
- integration-test - executa testes adicionais, geralmente requerem que a aplicação já esteja “empacotada”
- verify - verifica se o “package” é válido
- install - instala o pacote dentro do repositório local (pasta .m2 na pasta do usuário no sistema operacional - /home/[usuario] no Linux e c:\users\[usuario] no Windows)
- deploy - realiza a “disponibilização” da aplicação em um servidor remoto (aplicação) ou repositório (biblioteca)
- clean - apaga os pacotes gerados e os códigos compilados

# Maven - Como Usar



## Instalado

Maven é uma aplicação em java que depende de uma versão do java instalada e a variável de ambiente JAVA\_HOME definida. Para instalá-la, basta fazer o download no site <http://maven.apache.org/download.cgi> e seguir as instruções na página <http://maven.apache.org/install.html>

Depois de instalada, deve-se usar o comando mvn na pasta do projeto, informando os parâmetros (goals, profile, etc):

```
mvn clean compile
```

```
mvn package
```

# Maven - Como Usar



## Wrapper

O Maven também poderá ser utilizado por meio do Maven Wrapper. Ele permite incluir uma versão específica do maven dentro do próprio projeto, não sendo necessário que ele esteja instalado na máquina para a sua utilização.

Na pasta do projeto são incluídos arquivos de script para cada sistema operacional, além de uma pasta com os jars necessários para a sua execução.

No windows, o script se chama mvnw.cmd e no linux mvnw, e para sua execução em cada sistema (estando na pasta do projeto):

Windows: `mvnw.cmd clean package`

Linux e Mac: `./mvnw clean package`

**IMPORTANTE:** a maioria das IDEs já tem suporte a desenvolver aplicações em maven, não sendo necessário, na maioria das vezes, a execução manual destes comandos, mas é bom saber de sua existência e como executá-los

# Maven + Spring Boot

Hello World  
Honua

Aloha

Ola Mundo  
にちは世界

こん

Hola Mundo  
mundo

salve

Bonjour le monde

Привет, мир

# Olá Mundo - Spring Boot



Maneira rápida de criar um projeto maven com spring-boot é através do site spring inicializr <https://start.spring.io/>

The screenshot shows the Spring Initializr web interface. It is divided into several sections for configuring a new project:

- Project:** Includes radio buttons for **Maven Project** (selected), **Gradle Project**, and **Language** options: **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions: **3.0.0 (SNAPSHOT)**, **3.0.0 (M4)**, **2.7.4 (SNAPSHOT)**, **2.7.3** (selected), **2.6.12 (SNAPSHOT)**, and **2.6.11**.
- Project Metadata:** A series of text input fields for:
  - Group:** com.example
  - Artifact:** demo
  - Name:** demo
  - Description:** Demo project for Spring Boot
  - Package name:** com.example.demo
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions: **18**, **17** (selected), **11**, and **8**.
- Dependencies:** A section on the right with the text "No dependency selected" and a button labeled "ADD DEPENDENCIES... CTRL + B".

At the bottom of the interface, there are three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**

# Olá Mundo - Spring Boot



Iremos preencher os dados do projeto:

- **Project** - Maven Project
- **Language** - Java
- **Spring Boot** - 2.7.2
- **groupId** – br.org.serratec
- **artifactId** – ola-mundo
- **name** – ola-mundo
- **Description** - Olá Mundo
- **Package name** - br.org.serratec.olamundo
- **packaging** – jar

**Project**  
☒ Maven Project ☐ Gradle Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M4) ☐ 2.7.3 (SNAPSHOT) ☒ 2.7.2  
☐ 2.6.11 (SNAPSHOT) ☐ 2.6.10

**Project Metadata**  

Group

br.org.serratec

Artifact

ola-mundo

Name

ola-mundo

Description

Olá Mundo

Package name

br.org.serratec.olamundo

Packaging

☒ Jar ☐ War

Java

☐ 18 ☐ 17 ☒ 11 ☐ 8

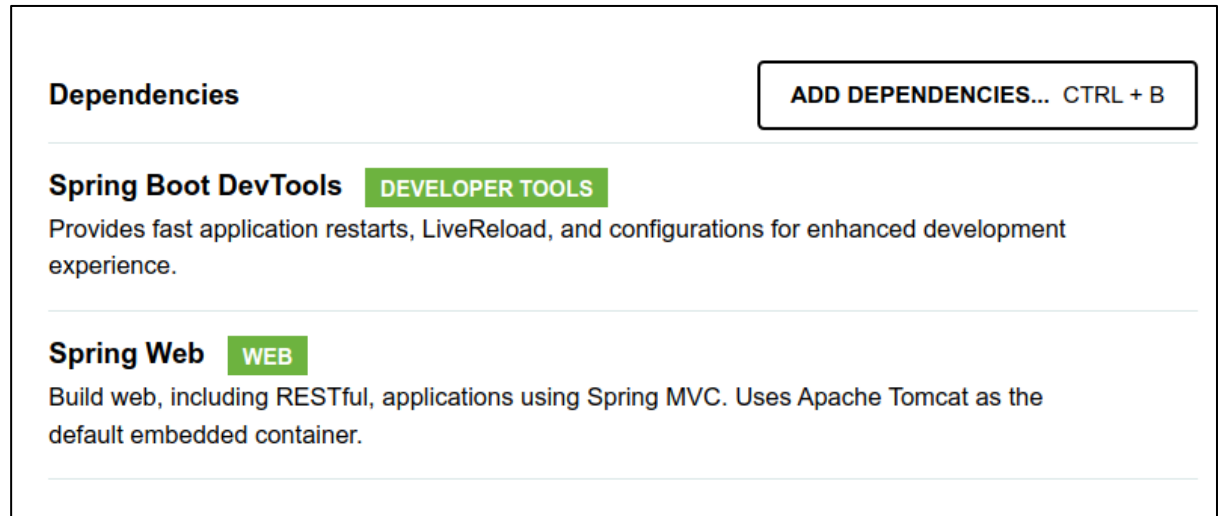


# Olá Mundo - Spring Boot



Vamos incluir as dependências:

- Spring Boot DevTools - auxilia no desenvolvimento da aplicação, reiniciando-a automaticamente quando alguma alteração é feita no código
- Spring Web - permite criar serviço web seguindo o protocolo REST, usando o servidor Apache Tomcat “embutido” na aplicação



# Olá Mundo - Spring Boot



Ao clicarmos no botão explorer, podemos ter um preview do projeto, sua estrutura de pastas e o conteúdo dos arquivos:

- .gitignore - arquivo contendo lista de arquivos a serem ignorados pelo git
- .mvn - pasta contendo maven wrapper (jar e configurações)
- HELP.md - “documento” contendo links sobre maven, spring boot, etc.
- mvnw - script maven wrapper para linux e mac
- mvnw.cmd - script maven wrapper para windows
- pom.xml - arquivo de configuração do projeto
- src - pasta contendo fontes do projeto
  - src/main/java - códigos em java
  - src/main/resources - arquivos de recurso: configuração, imagens, páginas, etc.
  - src/test/java - testes unitários

The screenshot displays a file explorer interface for a project named 'ola-mundo.zip'. On the left, a tree view shows the project structure: .gitignore, .mvn, HELP.md, mvnw, mvnw.cmd, pom.xml, src (main, java, br, org, serratec, olamundo, resources, test), and test. The 'pom.xml' file is selected, and its content is shown on the right. The XML content defines a Spring Boot project with dependencies on spring-boot-starter-parent, spring-boot-starter-web, and spring-boot-devtools. At the top right, there are 'DOWNLOAD' and 'COPY' buttons. At the bottom, there are 'DOWNLOAD CTRL + d' and 'CLOSE ESC' buttons.

```
<?xml version="1.0" encoding="UTF-8"?>
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>br.org.serratec</groupId>
  <artifactId>ola-mundo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ola-mundo</name>
  <description>Olá Mundo</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
  </dependencies>
</project>
```

# Olá Mundo - Spring Boot



No POM.XML é possível identificar as informações preenchidas no formulário e o bloco parent indica que nosso projeto “herda” configurações do projeto spring-boot-starter-parent:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.2</version>
    <relativePath/> <!-- Lookup parent from repository -->
  </parent>
  <groupId>br.org.serratec</groupId>
  <artifactId>ola-mundo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ola-mundo</name>
  <description>Olá Mundo</description>
  <properties>
    <java.version>11</java.version>
  </properties>
```

Project	Language
<input checked="" type="radio"/> Maven Project <input type="radio"/> Gradle Project	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin <input type="radio"/> Groovy
<b>Spring Boot</b>	
<input type="radio"/> 3.0.0 (SNAPSHOT) <input type="radio"/> 3.0.0 (M4) <input type="radio"/> 2.7.3 (SNAPSHOT) <input checked="" type="radio"/> 2.7.2	
<input type="radio"/> 2.6.11 (SNAPSHOT) <input type="radio"/> 2.6.10	
<b>Project Metadata</b>	
Group	br.org.serratec
Artifact	ola-mundo
Name	ola-mundo
Description	Olá Mundo
Package name	br.org.serratec.ola-mundo
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 18 <input type="radio"/> 17 <input checked="" type="radio"/> 11 <input type="radio"/> 8

# Olá Mundo - Spring Boot



Também é possível identificar as dependências adicionadas ao projeto.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

## Dependencies

ADD DEPENDENCIES... CTRL + B

### Spring Boot DevTools

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

### Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

# Olá Mundo - Spring Boot



O projeto já vem com uma classe contendo um método **main**, que é responsável por iniciar nossa aplicação web.

```
package br.org.serratec.olamundo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OlaMundoApplication {

    public static void main(String[] args) {

        SpringApplication.run(OlaMundoApplication.class, args);

    }

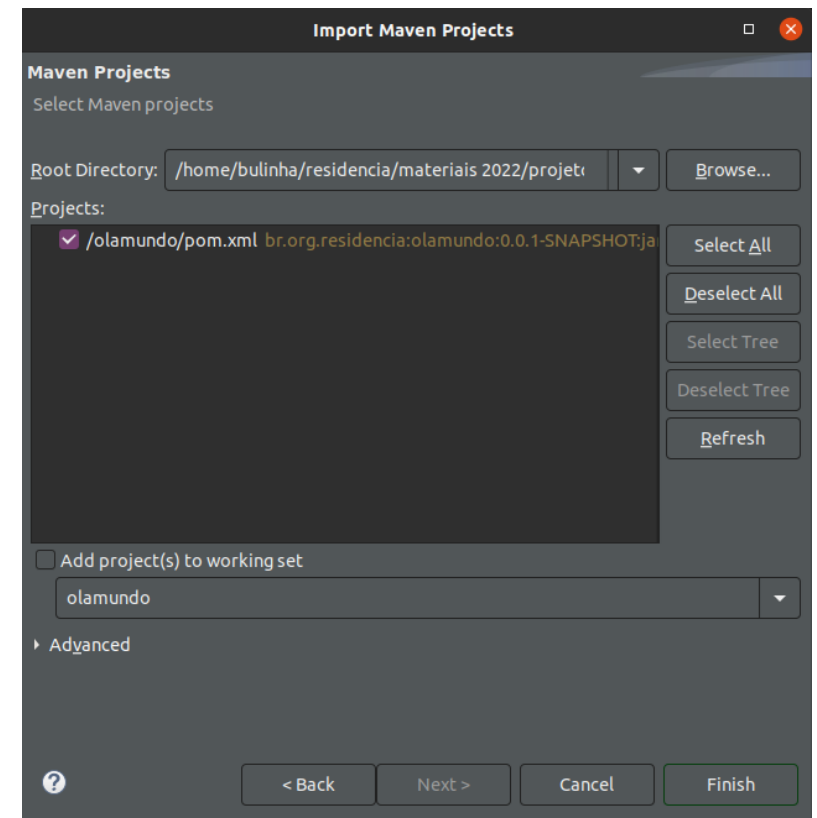
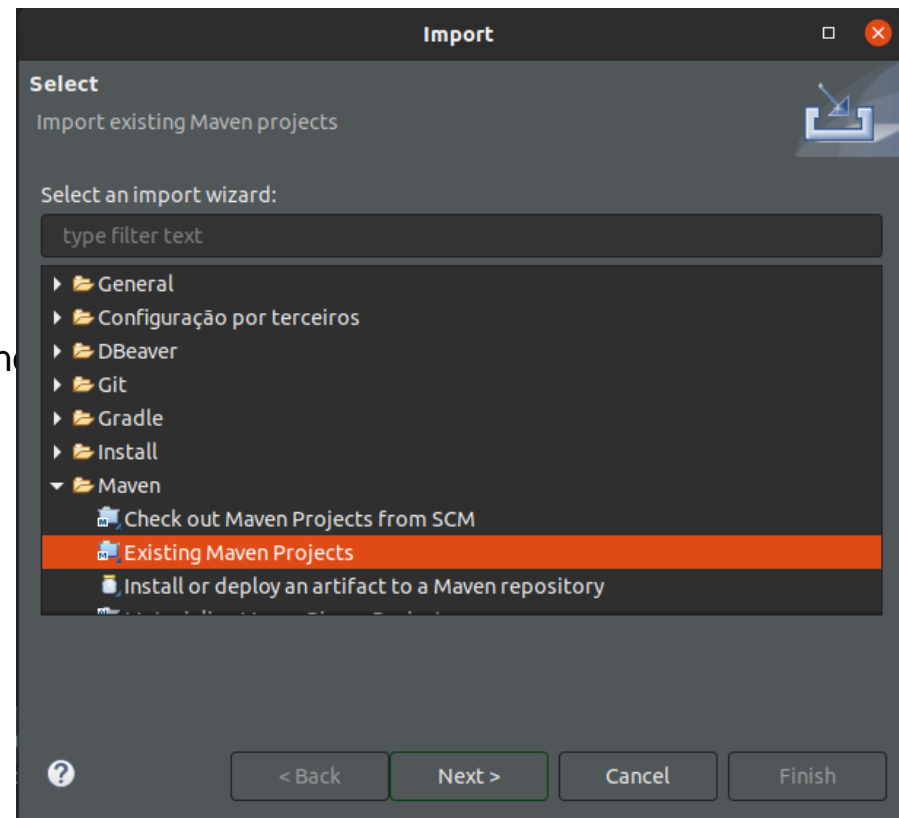
}
```

# Olá Mundo - Spring Boot



Após realizar o download do projeto, basta descompactá-lo em uma pasta e abri-lo na sua IDE para executá-lo. Nas IDEs VS Code (com as extensões para java e spring boot) e IntelliJ, basta abrir a pasta do projeto. Para a IDE Eclipse, faz-se necessário realizar a importação do projeto:

1. Acessar o menu File > Import
2. Selecionar Existing Maven Projects
3. Clicar no botão Next
4. Clicar selecionar projeto
5. Selecionar o pom.xml do projeto
6. Clicar



# Olá Mundo - Spring Boot



Vamos criar nosso primeiro Controller. Para isso, devemos criar uma classe chamada **OlaMundoController**, como no exemplo abaixo, incluindo as anotações **RestController**, que é um método que retorna uma string com a anotação **GetMapping**

```
package br.org.residencia.olamundo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class OlaMundoController {
    @GetMapping
    public String getOlaMundo() {
        return "Ola mundo";
    }
}
```

# Olá Mundo - Spring Boot



Execute a classe `OlaMundoApplication`. No console será possível ver o “log” da aplicação com diversas informações, incluindo a porta onde ela estará sendo executada. Lembre-se que por padrão, as aplicações web java são executadas na porta 8080.

```
<terminated> /usr/lib/jvm/java-11-openjdk-amd64/bin/java (14 de ago de 2022 11:25:21) [pid: 455361]

  ____ _
 / ___ \| | | |
/ /___ \| |_| |
\___)___|_|_|_|
:: Spring Boot :: (v2.7.0)

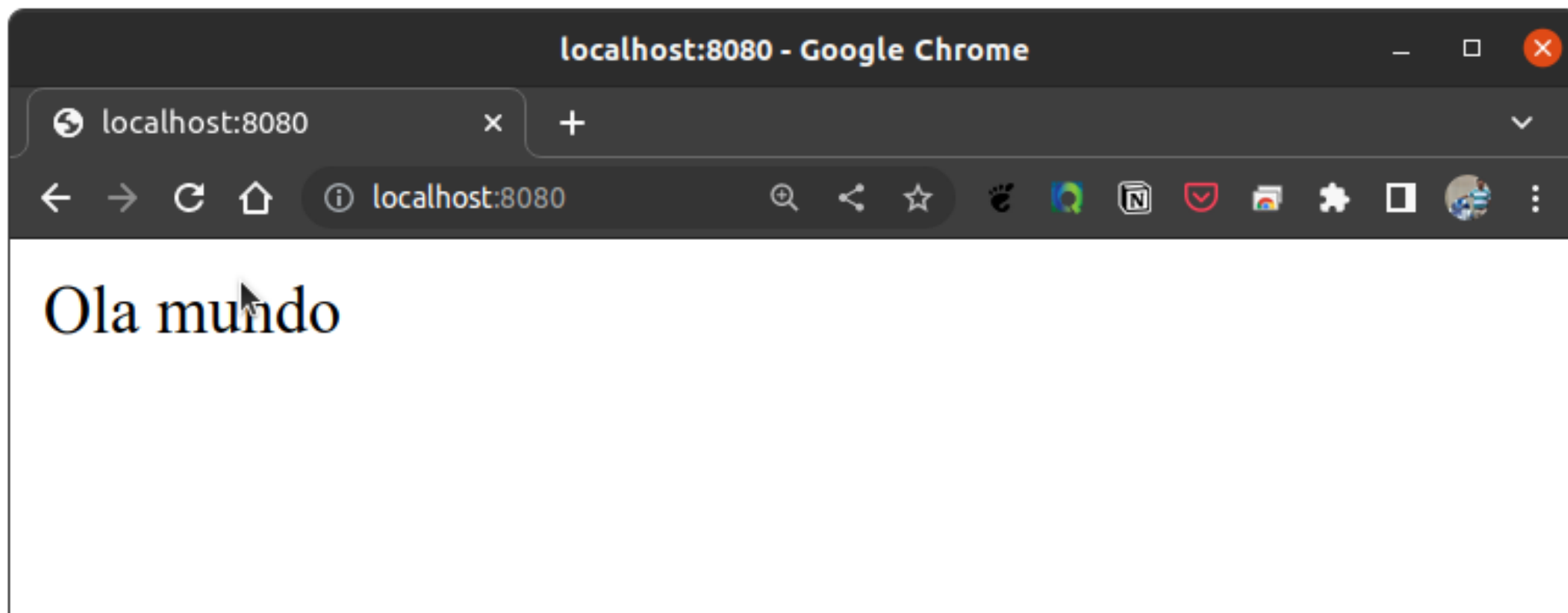
2022-08-14 11:25:22.290 INFO 455361 --- [main] b.o.r.olamundo.OlamundoApplication : Starting OlamundoApplication using Java 11.0.1
2022-08-14 11:25:22.293 INFO 455361 --- [main] b.o.r.olamundo.OlamundoApplication : No active profile set, falling back to 1 default profile: 'default'
2022-08-14 11:25:22.993 INFO 455361 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-08-14 11:25:23.001 INFO 455361 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-14 11:25:23.002 INFO 455361 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.63]
2022-08-14 11:25:23.074 INFO 455361 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-14 11:25:23.074 INFO 455361 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2022-08-14 11:25:23.375 INFO 455361 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/'
2022-08-14 11:25:23.385 INFO 455361 --- [main] b.o.r.olamundo.OlamundoApplication : Started OlamundoApplication in 1.362 seconds (JVM running for 1.471)
2022-08-14 11:25:25.322 INFO 455361 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-08-14 11:25:25.323 INFO 455361 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-14 11:25:25.323 INFO 455361 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
2022-08-14 11:53:14.343 INFO 455361 --- [n(38)-127.0.0.1] inMXBeanRegistrar$SpringApplicationAdmin : Application shutdown requested.
```



# Olá Mundo - Spring Boot



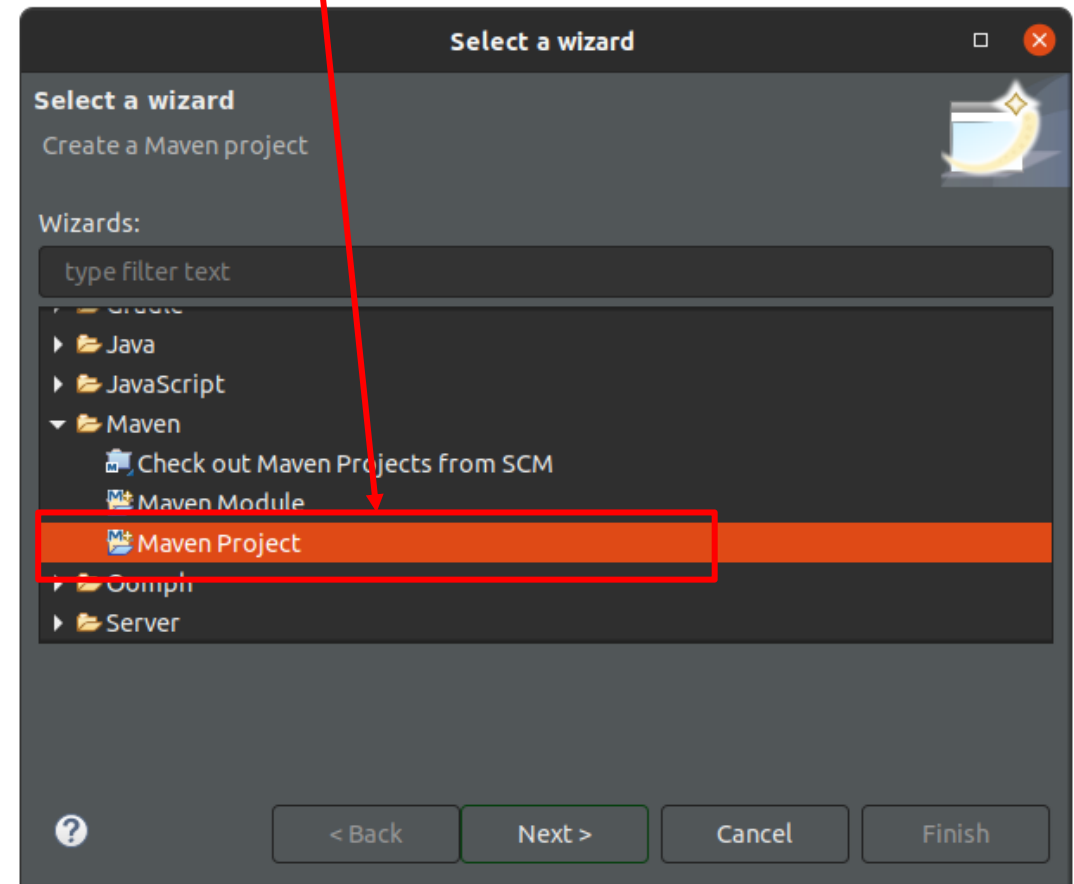
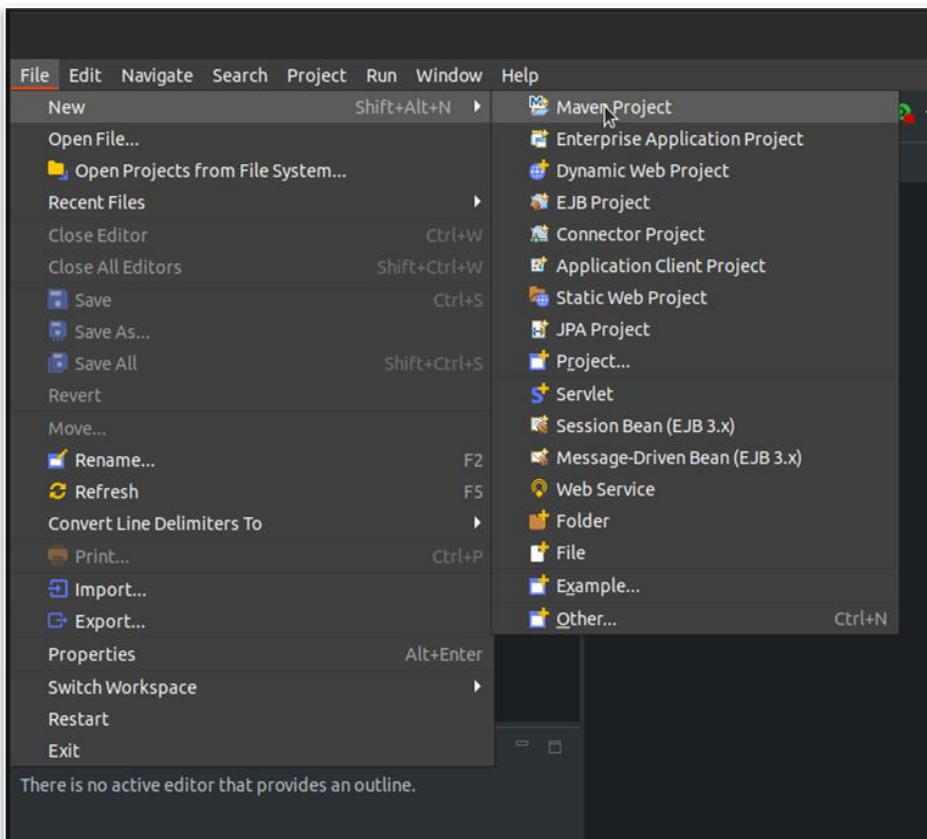
Se usarmos o navegador para acessar o endereço <http://localhost:8080>, poderemos ver a mensagem a seguir:



# Projeto Spring Boot a partir do Maven



Selecione o menu **File > New > Maven Project**, ou **File > New > Others**, selecionando **Maven Project** na lista de Wizards e clique em **Next**



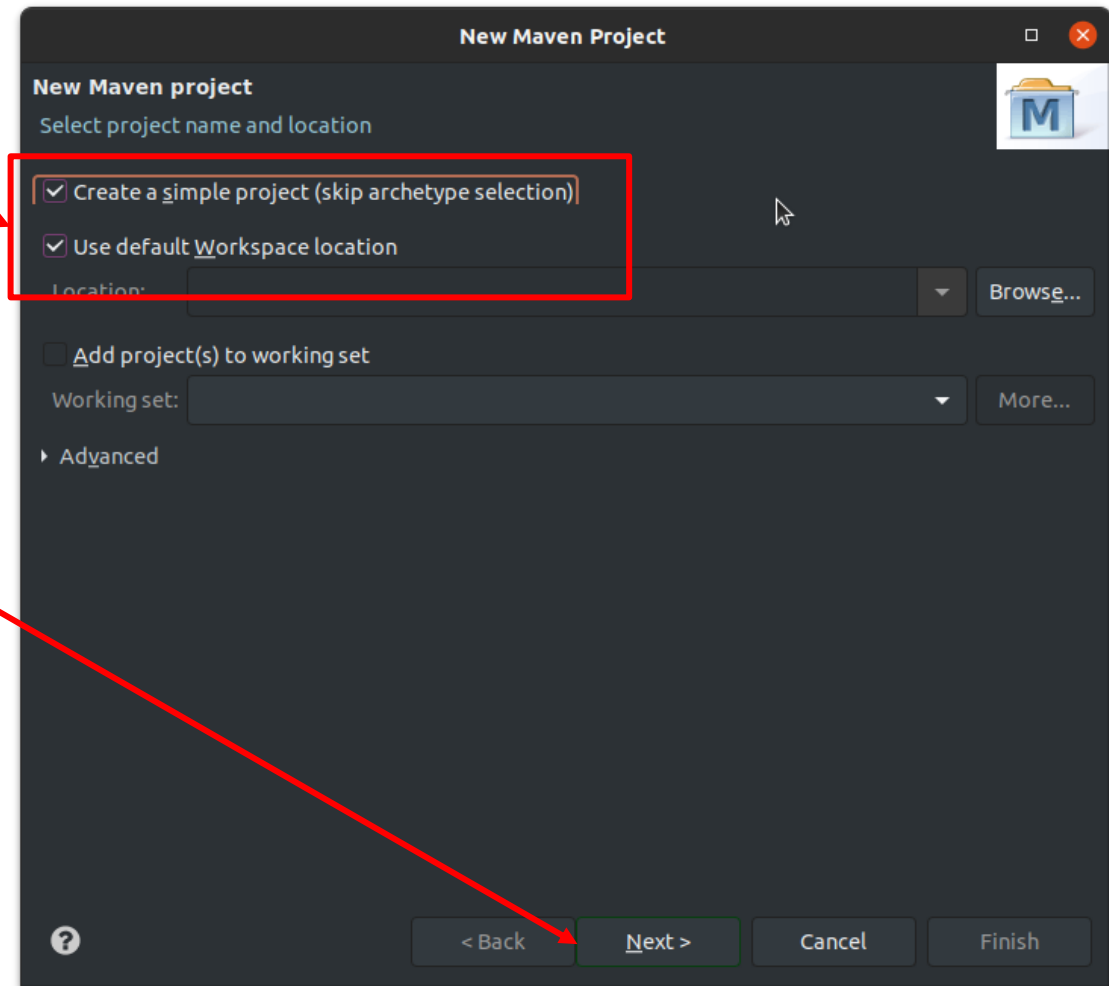
# Criando projeto Maven



Na janela de New Maven Project selecionar as opções

- **Create a simple project (skip archetype selection)**
- **Use default workspace location**

Clicar no botão **Next**



# Projeto Spring Boot a partir do Maven



Preencher a tela:

- **Group id:** identifica a qual grupo o projeto pertence. Pode ser como um nome de pacote que representa a empresa
  - *org.serratec.backend*
- **Artifact id:** nome do projeto. Vai compor o nome anterior
  - *projeto01*
- **Version:** versão do projeto
- **Packing:** como sera empacotado
- **Name:** Nome do projeto
  - *Projeto 01 de backend*
- **Description:** descrição do projeto
  - *Projeto de exemplo com maven e spring boot*
- **Parent Project:** é possível “herdar” configurações de outro projeto

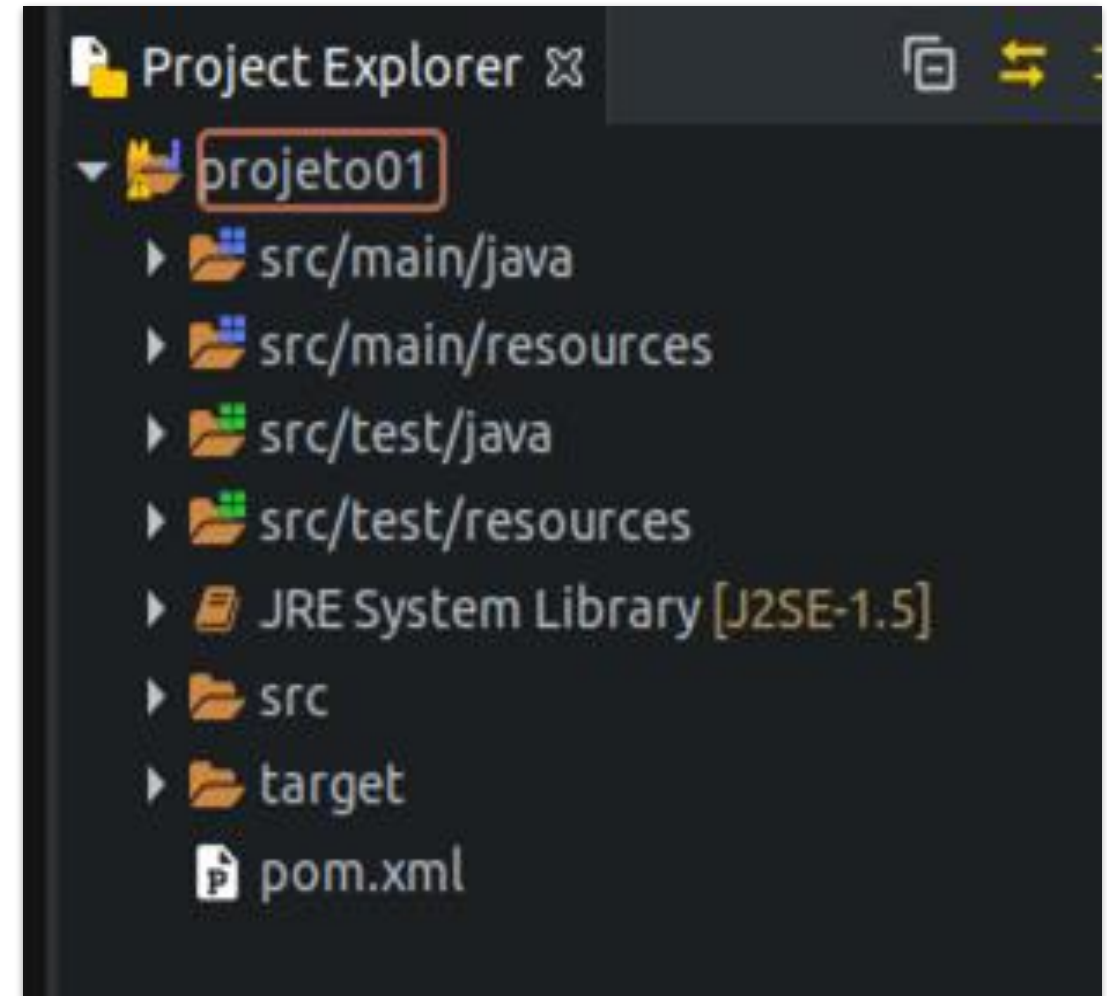
Clicar em Finish

# Projeto Spring Boot a partir do Maven



## Pastas de fontes (src)

- **src/main/java** - código java da aplicação
- **src/main/resources** - arquivos de recurso (imagens, configuração, etc...)
- **src/test/java** - código de teste unitário da aplicação
- **src/test/resources** - arquivos de recurso utilizado pelos testes
- **JRE System Library** - java que será utilizado por essa aplicação (não vem instalado, apenas está configurado por padrão)
- **target** - pasta onde irá ficar a aplicação empacotada (arquivo.jar)



# Arquivo POM.XML



Arquivo pom.xml com os dados que foi preenchido na tela de criação de projeto Maven



```
projeto01/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/xsi:schemaLocation"
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>org.serratec.backend</groupId>
4   <artifactId>projeto01</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <name>Projeto 01 de backend</name>
7   <description>Projeto de exemplo com maven e spring boot</description>
8 </project>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | **pom.xml**

Caso não abra diretamente nesta visualização, clique na aba “pom.xml”

# Convertendo p/ Spring Boot



Incluir este bloco no arquivo pom.xml (dentro da tag <project>)

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-
parent</artifactId>
  <version>2.7.3</version>
</parent>

<dependencies>
  <dependency>

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
web</artifactId>
  </dependency>
</dependencies>
```

Isso informa ao Maven que nosso projeto “herda” as configurações do projeto **spring-boot-starter-parent**.

E inclui a dependência do **spring-boot-starter-web**.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.serratec.backend</groupId>
  <artifactId>projeto01</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Projeto 01 de backend</name>
  <description>Projeto de exemplo com maven e spring boot</de

    <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.3.2.RELEASE</version>
    </parent>

    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
    </dependencies>

</project>
```

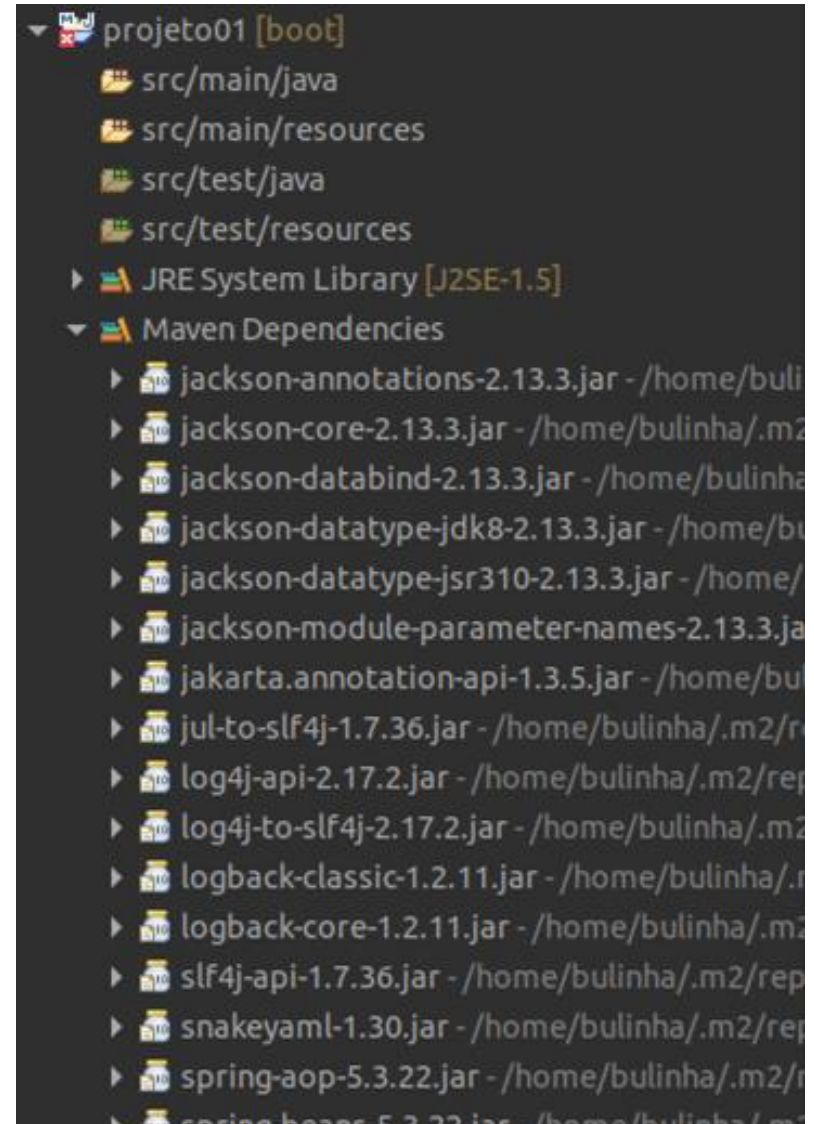


# Convertendo p/ Spring Boot



Ao salvar o projeto foi incluído a sessão Maven Dependencies na estrutura do projeto

- Nosso projeto “depende” do spring-boot-starter-web
- spring-boot-starter-web depende de spring-boot-starter
- spring-boot-starter depende de .....
- Maven é capaz de gerenciar todas as dependências hierarquicamente
- Cada um destes projetos tem seu próprio arquivo pom.xml

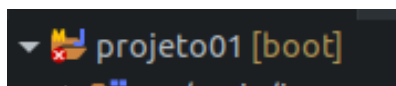




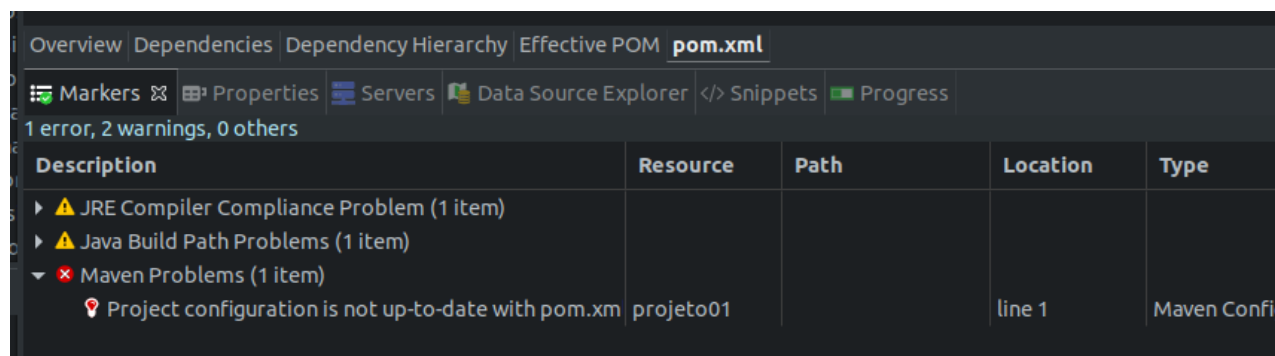
# Convertendo p/ Spring Boot



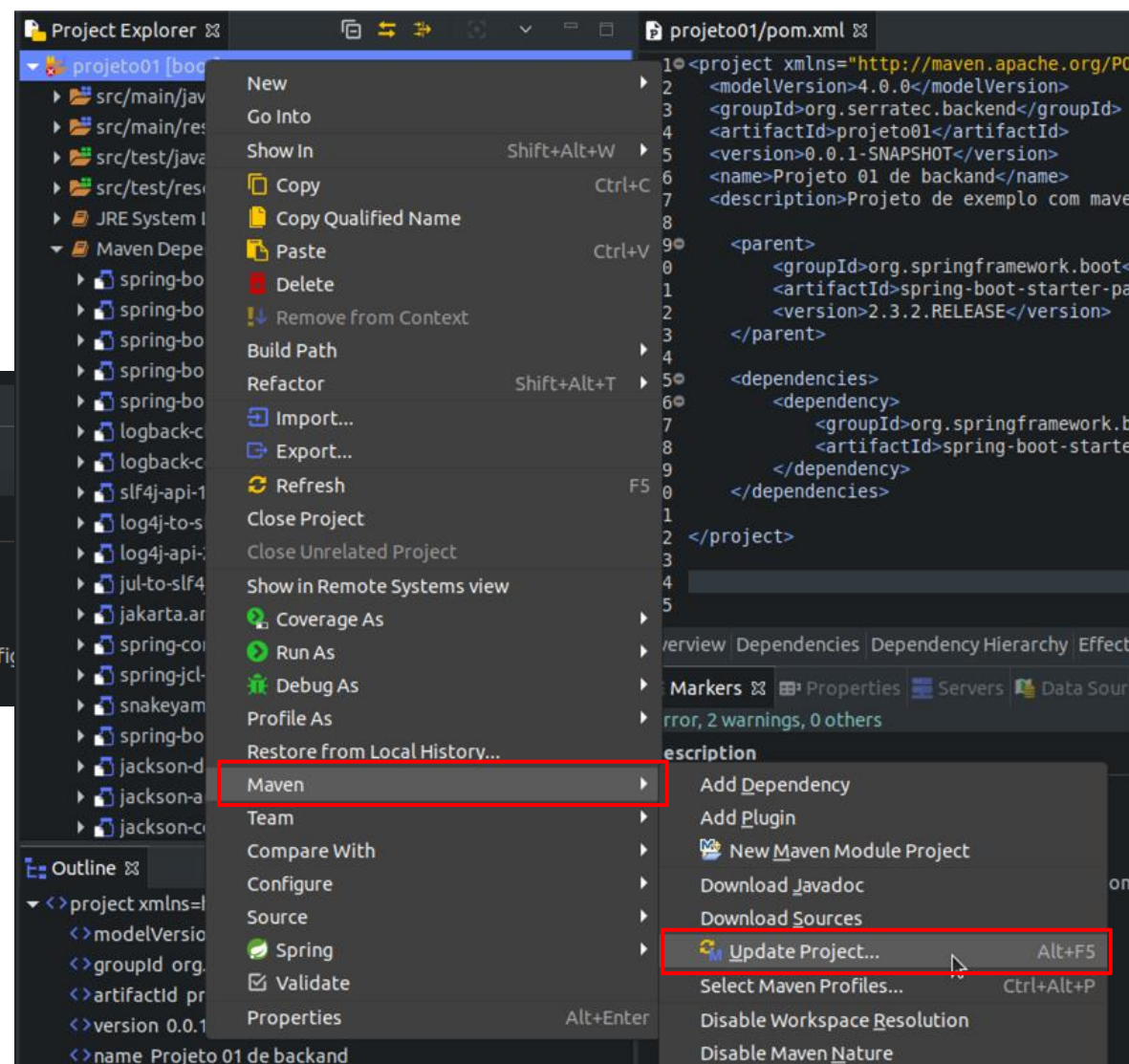
Ao salvar o projeto é possível perceber que há um erro sendo sinalizado no ícone no projeto



E na janela Markers é possível ver que o projeto não está atualizado.



Basta clicar com o botão direito do mouse no projeto e selecionar a opção Maven > Update Project... e clicar no botão Ok na janela aparecerá



# Convertendo p/ Spring Boot

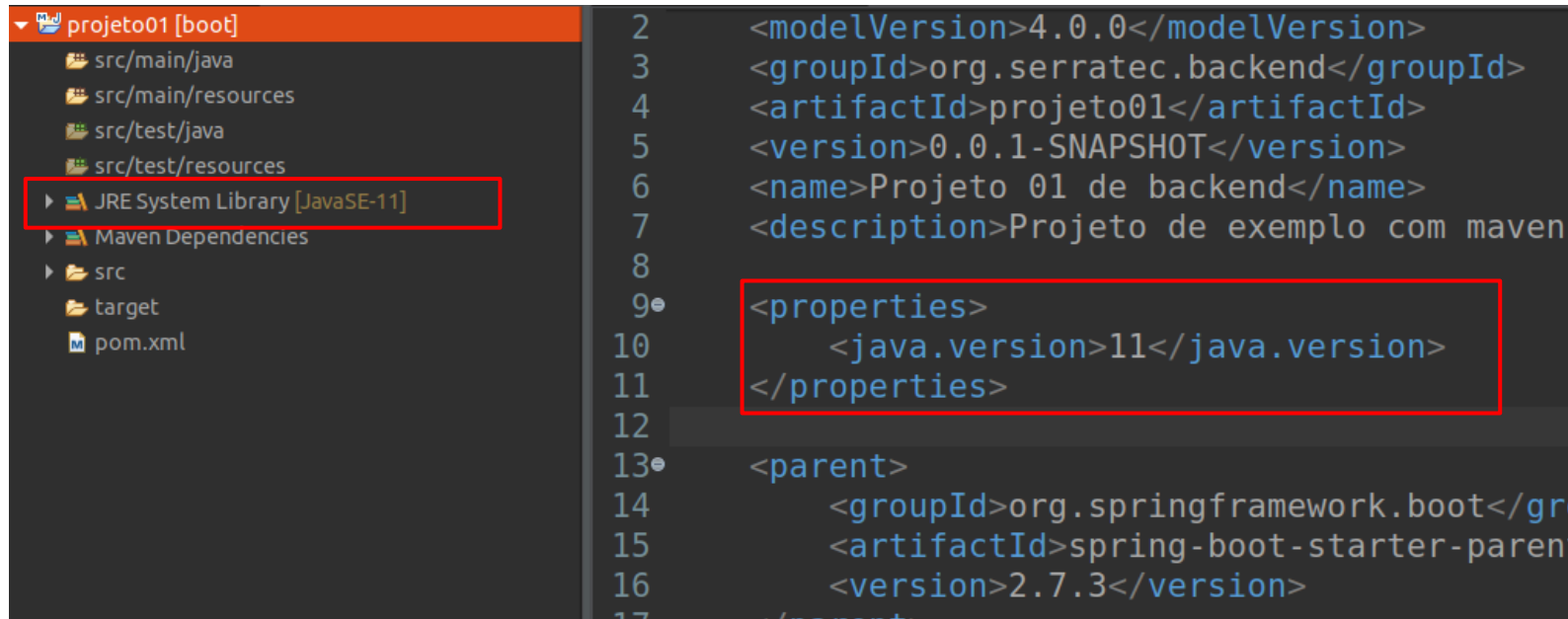


Passo opcional: caso a versão do java sendo exibida na estrutura do projeto não seja a 11

```
<properties>
```

```
<java.version>11</java.version>
```

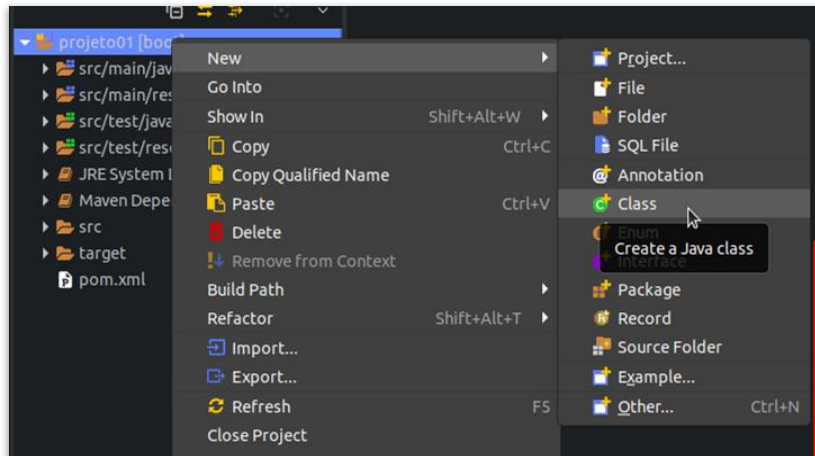
```
</properties>
```



# Criando Classe Main



Clique com o botão direito no projeto, New > Class

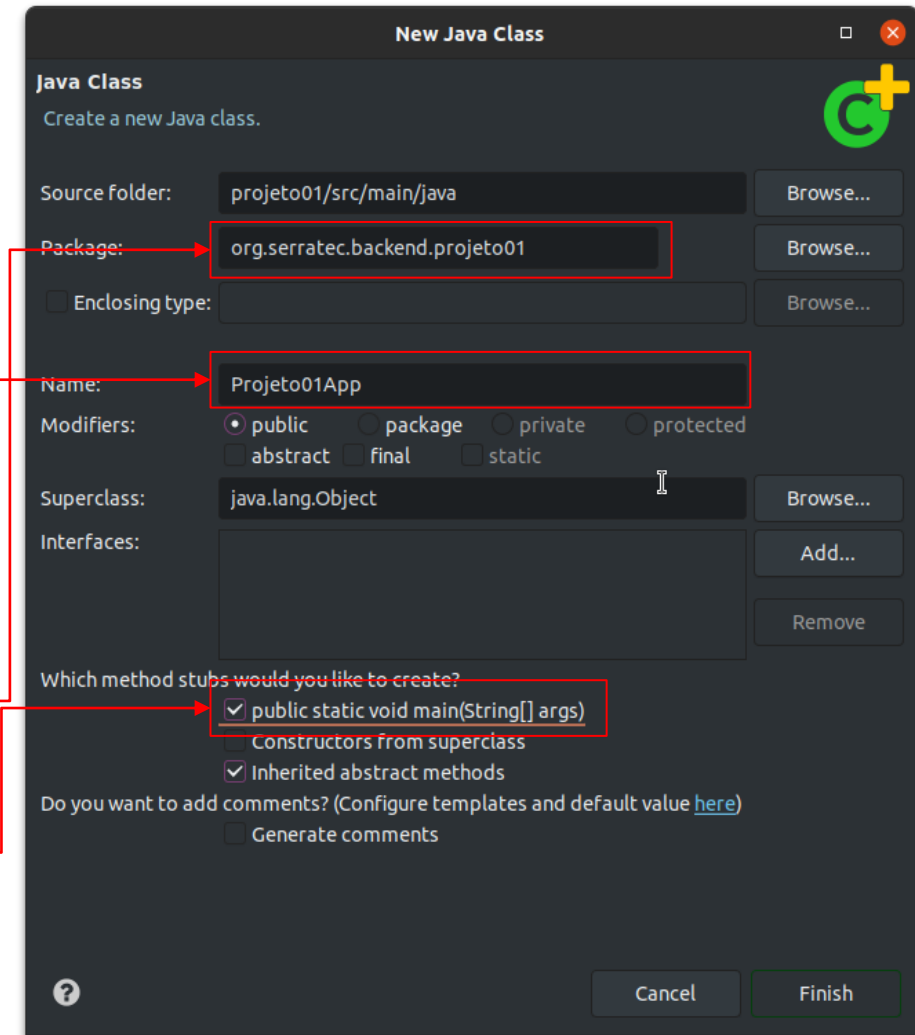


Preencha os campo

- Package: **org.serratec.backend.projeto01**
- Name: **Projeto01App**

E marque a opção “public static void main...”

Clique em Finish



# Criando Classe Main



- Incluir a anotação `@SpringBootApplication` na classe, ela indica que é uma aplicação SpringBoot com configuração padrão
- Incluir no método main a chamada ao método `SpringApplication.run(Classe, args)`, ele inicializa o SpringBoot, indicando qual a classe inicial que o ele irá utilizar e quais parâmetros ele recebera na inicialização
- Incluir os imports necessários (no eclipse basta digitar `ctrl+shift+o`)

Feito isso, nossa aplicação Spring Boot está pronta para iniciar

```
package projeto01;

public class Projeto01App {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }

}
```

```
package projeto01;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Projeto01App {

    public static void main(String[] args) {
        SpringApplication.run(Projeto01App.class, args);
    }

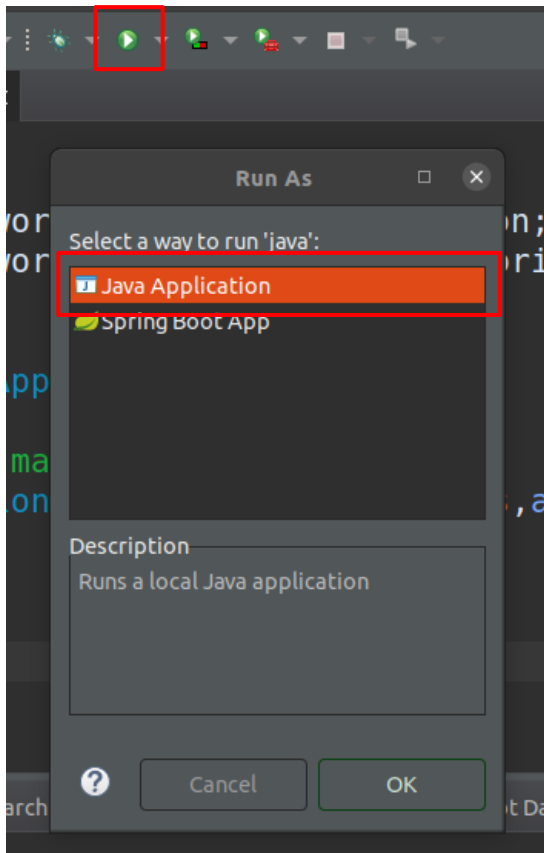
}
```

# Executando a Aplicação



1. Clicar no botão Run As...
2. Selecionar Java Application
3. Clicar em ok

A janela do Console deve ganhar o foco automaticamente e será possível perceber ao final do “Log”, que a aplicação iniciou (Started) em 1.4 segundos (no caso do exemplo). E também é possível ver que ela usa o Tomcat “embedded” e que pode ser acessado pela porta 8080



```
Problems Javadoc Declaration Search Console x Progress Debug Boot Dashboard Git Staging
Projeto01App [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (1 de set de 2022 12:55:18) [pid: 867932]

:: Spring Boot ::
(v2.7.3)

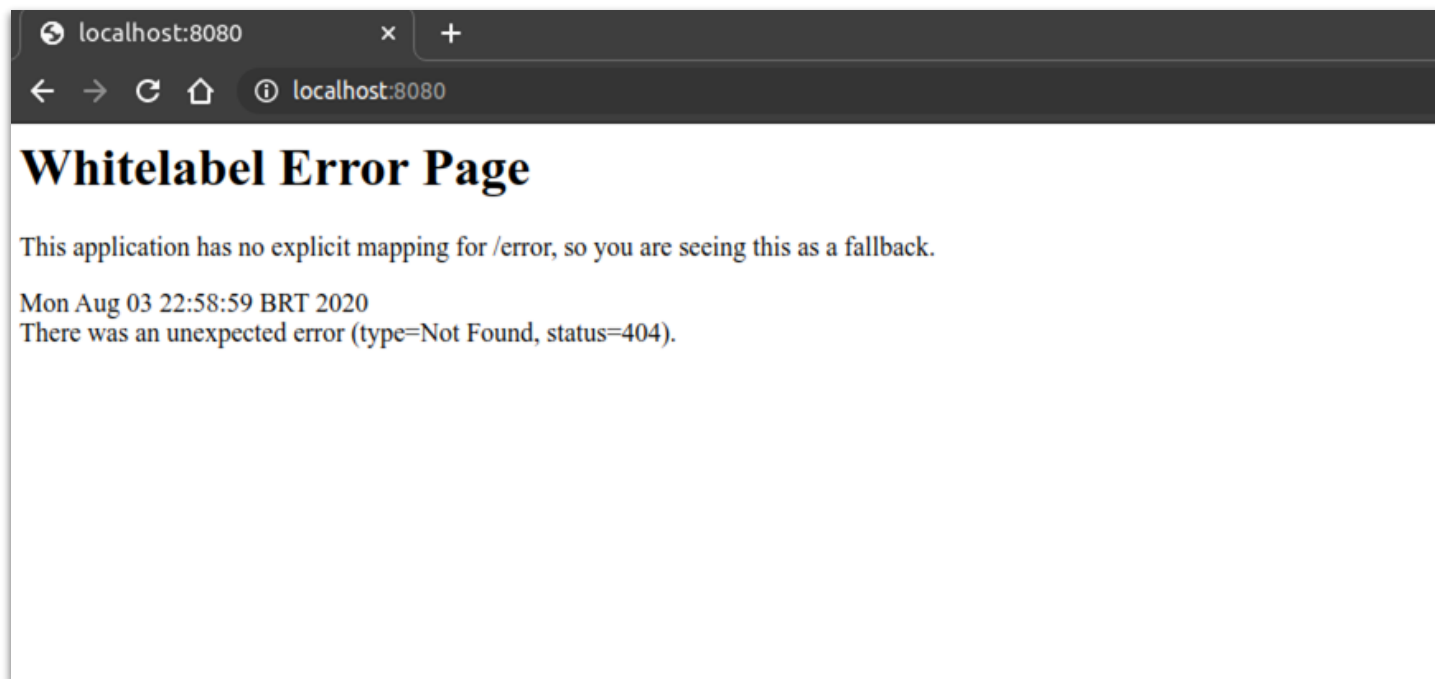
2022-09-01 12:55:22.409 INFO 867932 --- [main] projeto01.Projeto01App : Starting Projeto01App using Java 11.0.16 on bula-idk-notebook with P
2022-09-01 12:55:22.412 INFO 867932 --- [main] projeto01.Projeto01App : No active profile set, falling back to default profile: "default"
2022-09-01 12:55:26.508 INFO 867932 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-09-01 12:55:26.557 INFO 867932 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-09-01 12:55:26.558 INFO 867932 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-09-01 12:55:26.877 INFO 867932 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-09-01 12:55:26.877 INFO 867932 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4177 ms
2022-09-01 12:55:27.455 INFO 867932 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-09-01 12:55:27.464 INFO 867932 --- [main] projeto01.Projeto01App : Started Projeto01App in 5.912 seconds (JVM running for 7.839)
```

# Executando a Aplicação



Podemos verificar a aplicação na url: <http://localhost:8080>

Como não temos nada desenvolvido e nem mesmo uma página de erro configurada, o Spring Boot apresenta esta página padrão.



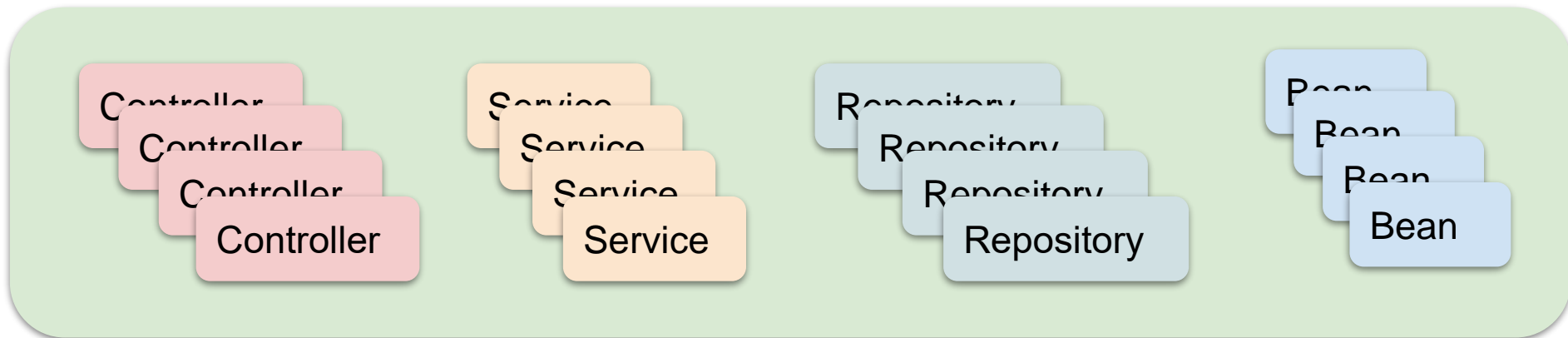
# Iniciando Spring Boot



```
SpringApplication.run(Projeto01App.class, args);
```

1. Inicia Configuração Padrão
2. Inicia o Spring Context
3. Realiza um “scan” de classpath
4. Inicia o Servidor Tomcat “embutido”

## Spring Context



*Importante: todo objeto dentro do contexto do Spring é considerado um **Bean***

# Criando o primeiro Controller



1. Criar uma nova classe chamada HelloWorldController
2. Adicionar um método OlaMundo que retorna uma String
3. Anotar a classe e o método com as seguintes anotações:
  - @RestController - indica que a classe será tratada pelo SpringBoot como um Controller
  - @RequestMapping("/ola") - associa o método anotado a url /ola
4. Acessar a url <http://localhost:8080/ola>  
Via browser e/ou via Postman

*Obs: @RequestMapping, por padrão, responde às chamadas do verbo/método GET*

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

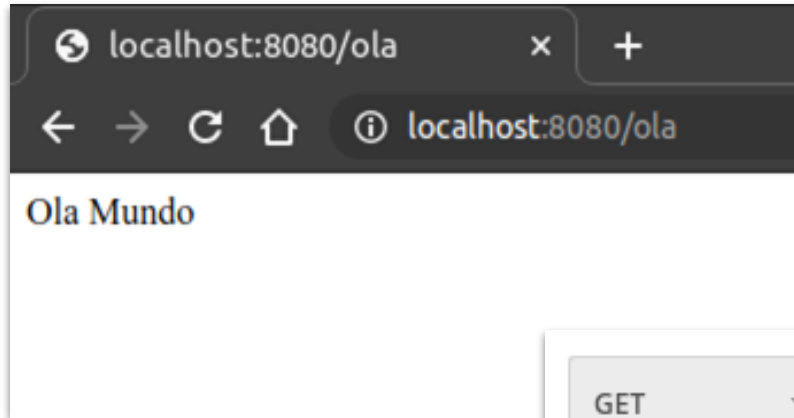
@RestController
public class HelloWorldController {

    @RequestMapping("/ola")
    public String OlaMundo() {
        return "Ola Mundo";
    }

}
```

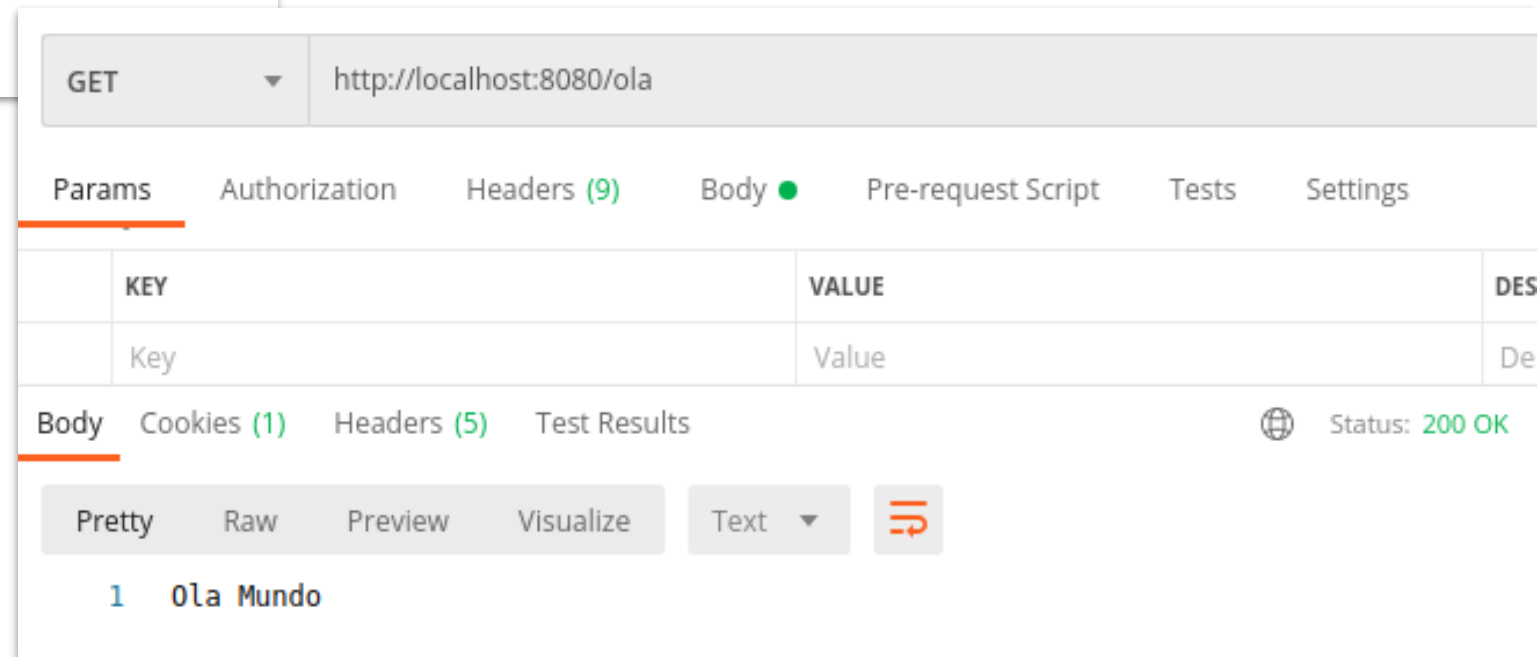


# Criando o primeiro Controller



```
~$ curl -X GET -i http://localhost:8080/ola
HTTP/1.1 200
Content-Type: text/plain; charset=UTF-8
Content-Length: 9
Date: Thu, 01 Sep 2022 17:03:03 GMT

Ola Mundo
```



# Criando o primeiro Controller



Adicionar mais um método conforme o exemplo e fazer o teste

```
@RestController
public class HelloWorldController {

    @RequestMapping("/ola")
    public String OlaMundo() {
        return "Ola Mundo";
    }

    @RequestMapping("/resposta")
    public String msg() {
        return "Oi";
    }
}
```

GET

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

Body Cookies Headers (5) Test Results 🌐 Status: 200 OK Time: 189 ms Size: 165 B

Pretty Raw Preview Visualize Text

1 Oi

# Criando o primeiro Controller



O **RequestMapping** possui algumas propriedades onde podemos definir o verbo, o path, o formato de saída e outras informações por exemplo.

```
@RestController
public class HelloWorldController {

    @RequestMapping("/ola")
    public String OlaMundo() {
        return "Ola Mundo";
    }

    @RequestMapping(value="/resposta", method = RequestMethod.GET, produces = { "application/json" })
    public String msg() {
        return "Oi";
    }

}
```

# Criando o primeiro Controller



Uma forma definir que é mais utilizada é inserindo a anotação `@RequestMapping` antes da definição da classe com o nome do recurso. Os verbos deverão ser definidos em cada método.

Vamos inserir uma nova classe com o nome `ExemplosController`

```
@RequestMapping("/api/v1")
public class ExemploController {

    @GetMapping
    public String teste(){
        return "Teste Serratec";
    }

}
```

```
@RequestMapping("/api/v1")
public class ExemploController {

    @GetMapping
    public String teste(){
        return "Teste Serratec";
    }

    @GetMapping("/oi")
    public String oi(){
        return "Oi";
    }

}
```

# Passagem de Parâmetros



Anotação `@RequestParam` associa um parâmetro do método java a um parâmetro da url.

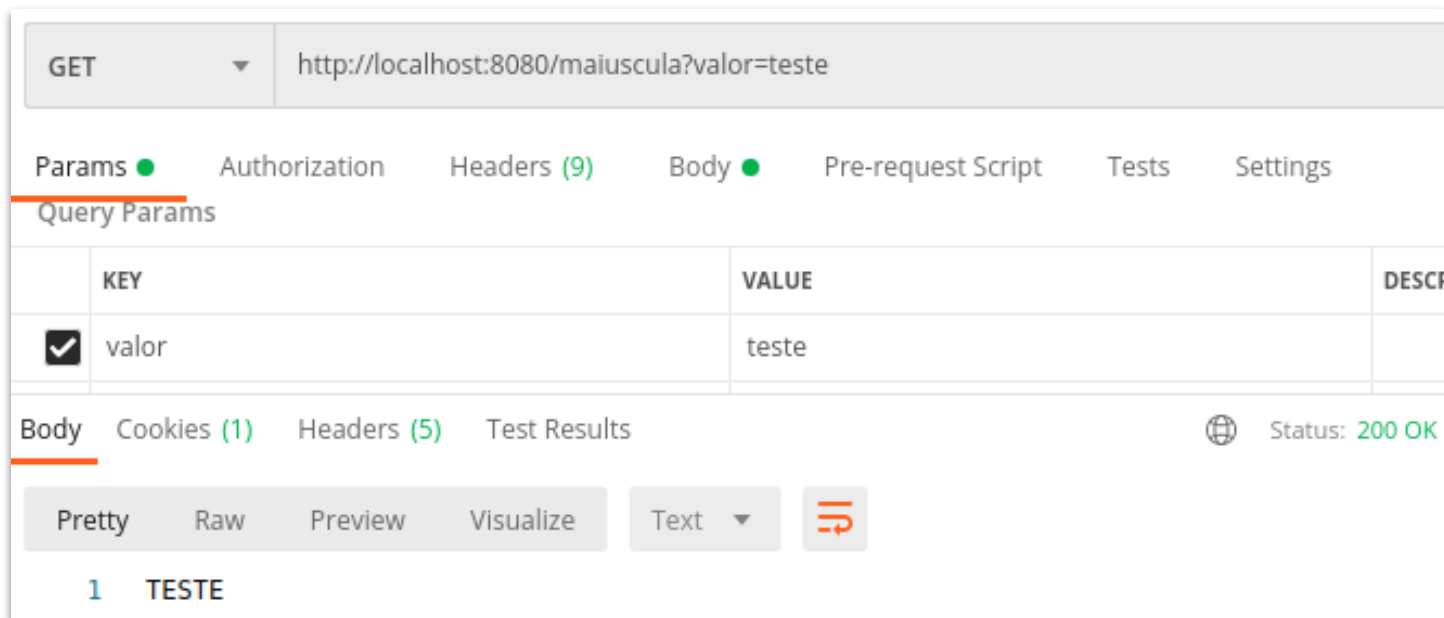
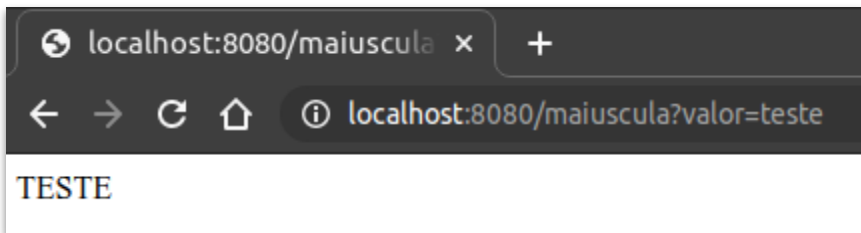
```
@GetMapping("/maiuscula")
public String maiuscula(@RequestParam String valor) {
    return valor.toUpperCase();
}
```

Parâmetros em URLs são passados ao final da url, após um ponto de interrogação “?” e com o formato **nome-parametro=valor** e separados por um e *comercial* “&”:

`http://localhost:8080/mapping?param1=val1&param2=val2&param3=val3`

É possível indicar um nome alternativo para o parâmetro da url que seja diferente do parâmetro da função:

`@RequestParam("nome-parametro")`

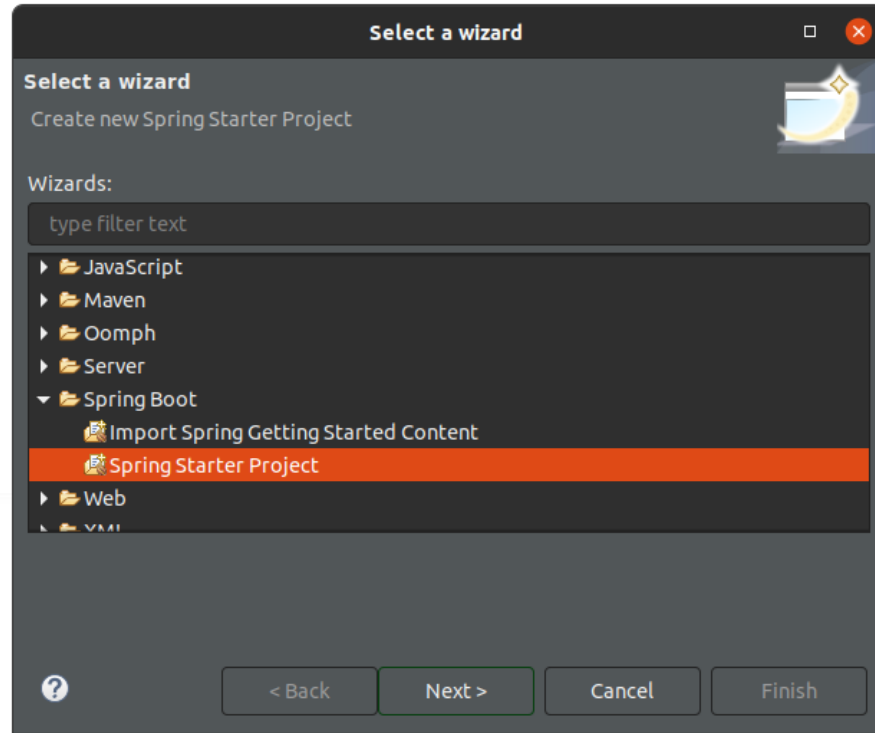


# Exercício



1. Implementar uma calculadora básica
  - Somar
  - Multiplicar
  - Dividir
  - Subtrair

# Projeto Spring Boot direto no Eclipse



- Disponível no Spring Tool Suite e no Eclipse com o plugin do Spring Tool
- Acesse o menu File > New > Other

# Novo Projeto Spring Boot



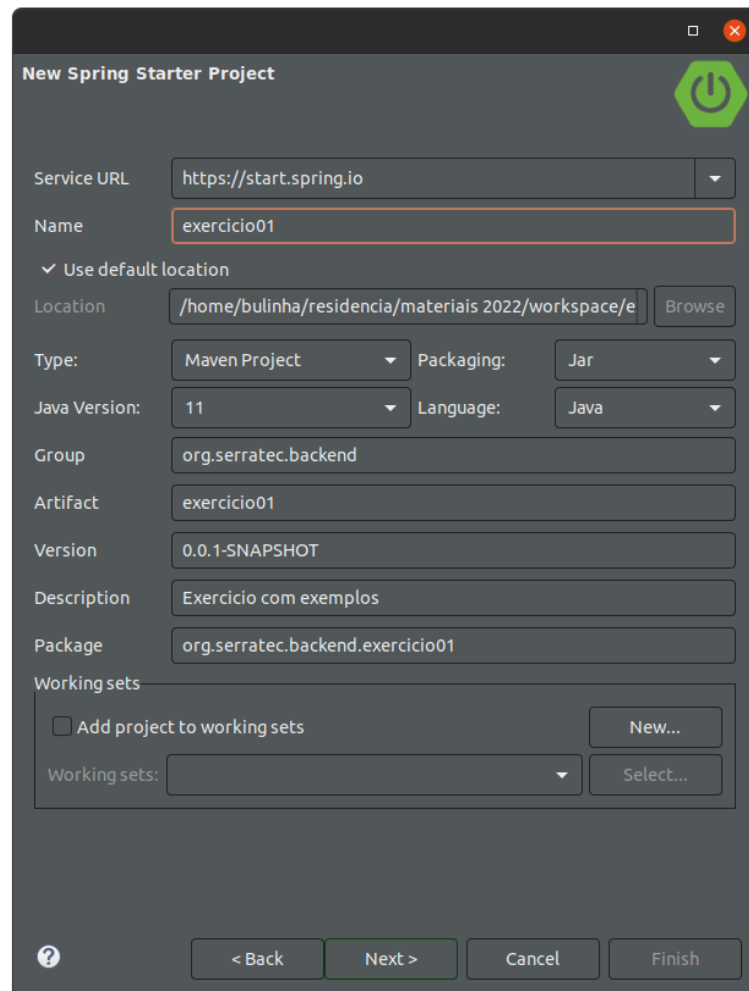
1) Preencha as informações e clique Next:

- **Name:** exercicio01
- **Group:** org.serratec.backend
- **Artifact:** exercicio01
- **Description:** Exercicio com exemplos
- **Package:** org.serratec.backend.exercicio01

1) Selecione a opção Spring Web em Frequently Used.

A opção web vai colocar as configurações necessárias no **pom.xml** para termos o servidor **TomCat** embarcado na aplicação.

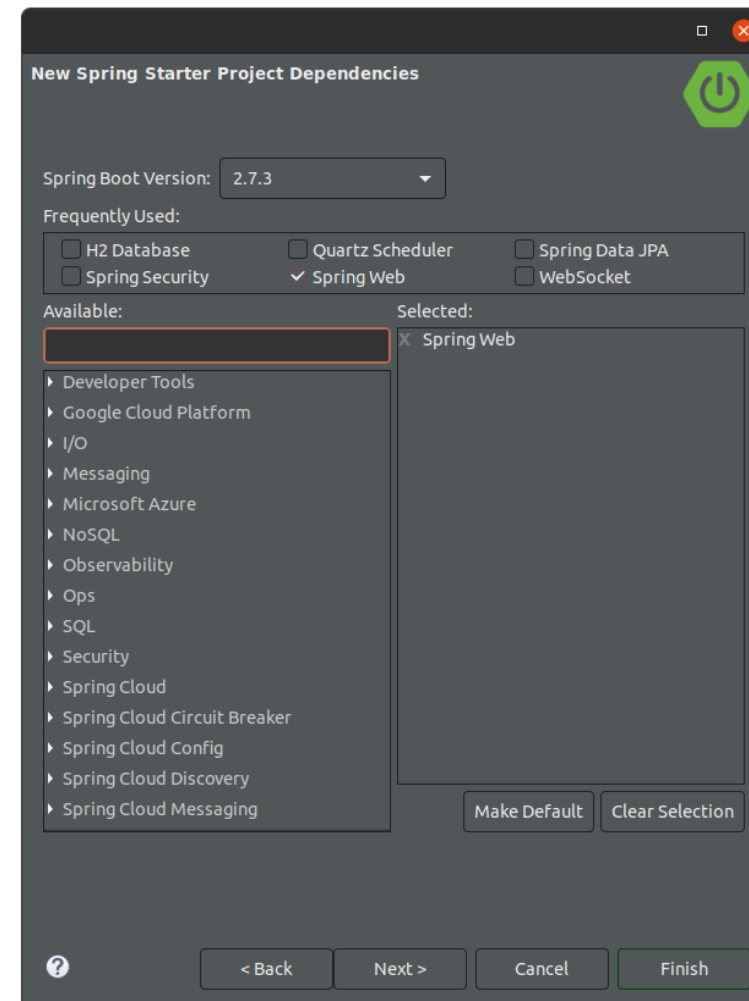
1) Clique em finish para encerrar



The 'New Spring Starter Project' dialog box is shown. It contains the following fields and options:

- Service URL: `https://start.spring.io`
- Name: `exercicio01`
- ☒ Use default location
- Location: `/home/bulinha/residencia/materiais 2022/workspace/e` (with a 'Browse' button)
- Type: `Maven Project` (dropdown), Packaging: `Jar` (dropdown)
- Java Version: `11` (dropdown), Language: `Java` (dropdown)
- Group: `org.serratec.backend`
- Artifact: `exercicio01`
- Version: `0.0.1-SNAPSHOT`
- Description: `Exercicio com exemplos`
- Package: `org.serratec.backend.exercicio01`
- Working sets: ☐ Add project to working sets (with 'New...' and 'Select...' buttons)

At the bottom, there are buttons: `< Back`, `Next >` (highlighted in green), `Cancel`, and `Finish`.



The 'New Spring Starter Project Dependencies' dialog box is shown. It contains the following fields and options:

- Spring Boot Version: `2.7.3` (dropdown)
- Frequently Used: ☐ H2 Database, ☐ Quartz Scheduler, ☐ Spring Data JPA, ☐ Spring Security, ☒ Spring Web, ☐ WebSocket
- Available: (list of categories like Developer Tools, Google Cloud Platform, etc.)
- Selected: `Spring Web` (with a close button 'X')
- Buttons: `Make Default` and `Clear Selection`

At the bottom, there are buttons: `< Back`, `Next >` (highlighted in green), `Cancel`, and `Finish`.



# Novo Projeto Spring Boot



“Herda” as configurações “padrão” do Spring

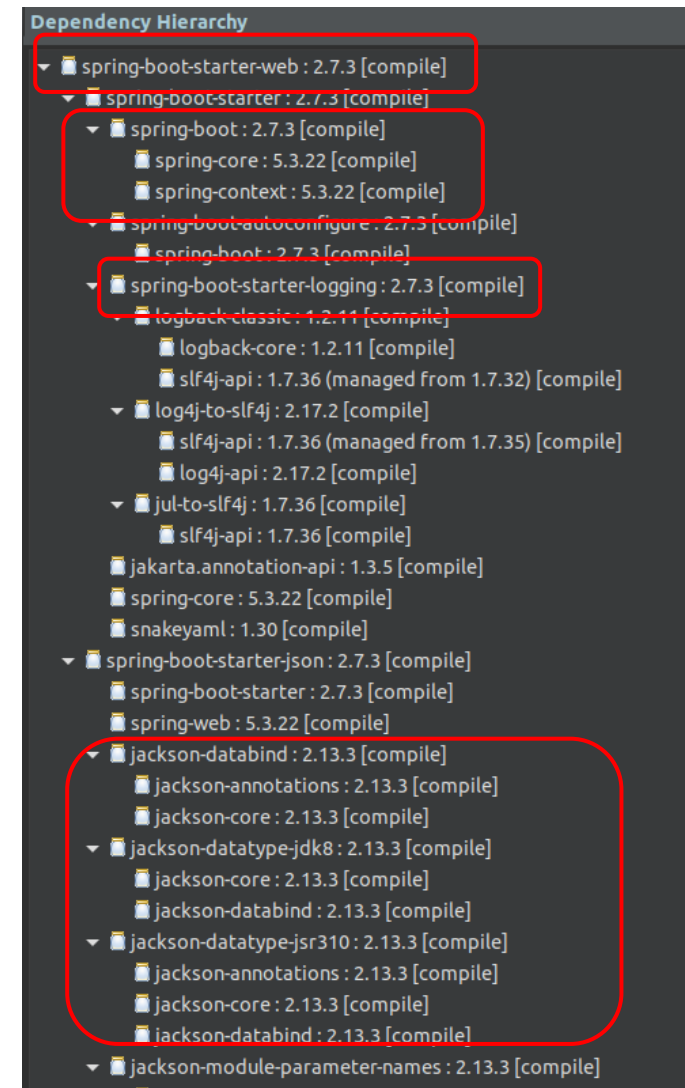
```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Inclui todas as dependências necessárias para web e suas configurações padrão

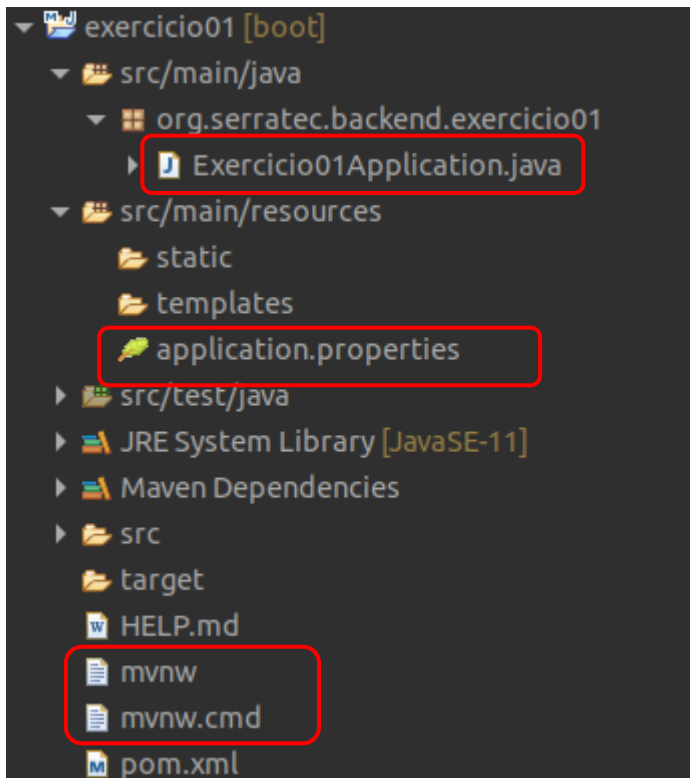
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Bibliotecas de Log, Json, Spring-Core, Spring-Context e Tomcat “Embedded” (embutido)

Tanto bibliotecas quanto suas versões são garantidas de funcionarem juntas por padrão - Esta é a grande vantagem do Spring Boot!



# Arquivos e Pastas do projeto



O maven também pode ser executado via linha de comando para fazer um build do projeto. O projeto já é criado com o maven wrapper, possuindo um arquivo **mvnw** que pode ser executado no Linux e o **mvnw.cmd** no Windows.

O arquivo **application.properties** é um arquivo de configuração onde serão inseridas diversas configurações como acesso a banco de dados por exemplo.

A classe **Exercicio01Application** foi criada automaticamente.

Selecione a classe e execute a aplicação.

# Domain



Criar a classe **Aluno** no pacote **domain**

**New Java Class**

Java Class  
Create a new Java class.

Source folder:  Browse...

Package:  Browse...

☐ Enclosing type:  Browse...

Name:

Modifiers:  
☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:  Browse...

Interfaces:  Add... Remove

Which method stubs would you like to create?  
☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Cancel Finish

```
public class Aluno {  
    private Long matricula;  
    private String nome;  
    private String telefone;  
    public Aluno(Long matricula, String nome, String telefone) {  
        super();  
        this.matricula = matricula;  
        this.nome = nome;  
        this.telefone = telefone;  
    }  
    public Aluno() {}  
    public Long getMatricula() {  
        return matricula;  
    }  
    public void setMatricula(Long matricula) {  
        this.matricula = matricula;  
    }  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getTelefone() {  
        return telefone;  
    }  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
}
```

# Controller



Criar a classe **AlunoController** no pacote **controller**

```
@RestController
@RequestMapping("/alunos")
public class AlunoController {
    private static List<Aluno> lista = new ArrayList<Aluno>();
    static {
        lista.add(new Aluno(2354L, "Carla", "2224-0439"));
        lista.add(new Aluno(2343L, "Carlos", "2334-0239"));
        lista.add(new Aluno(1409L, "Maria", "2343-2345"));
    }

    @GetMapping
    public List<Aluno> listar() {
        return lista;
    }
}
```

**@RestController** – Esta anotação informa que a classe será um controlador REST. Por padrão o retorno será em JSON.

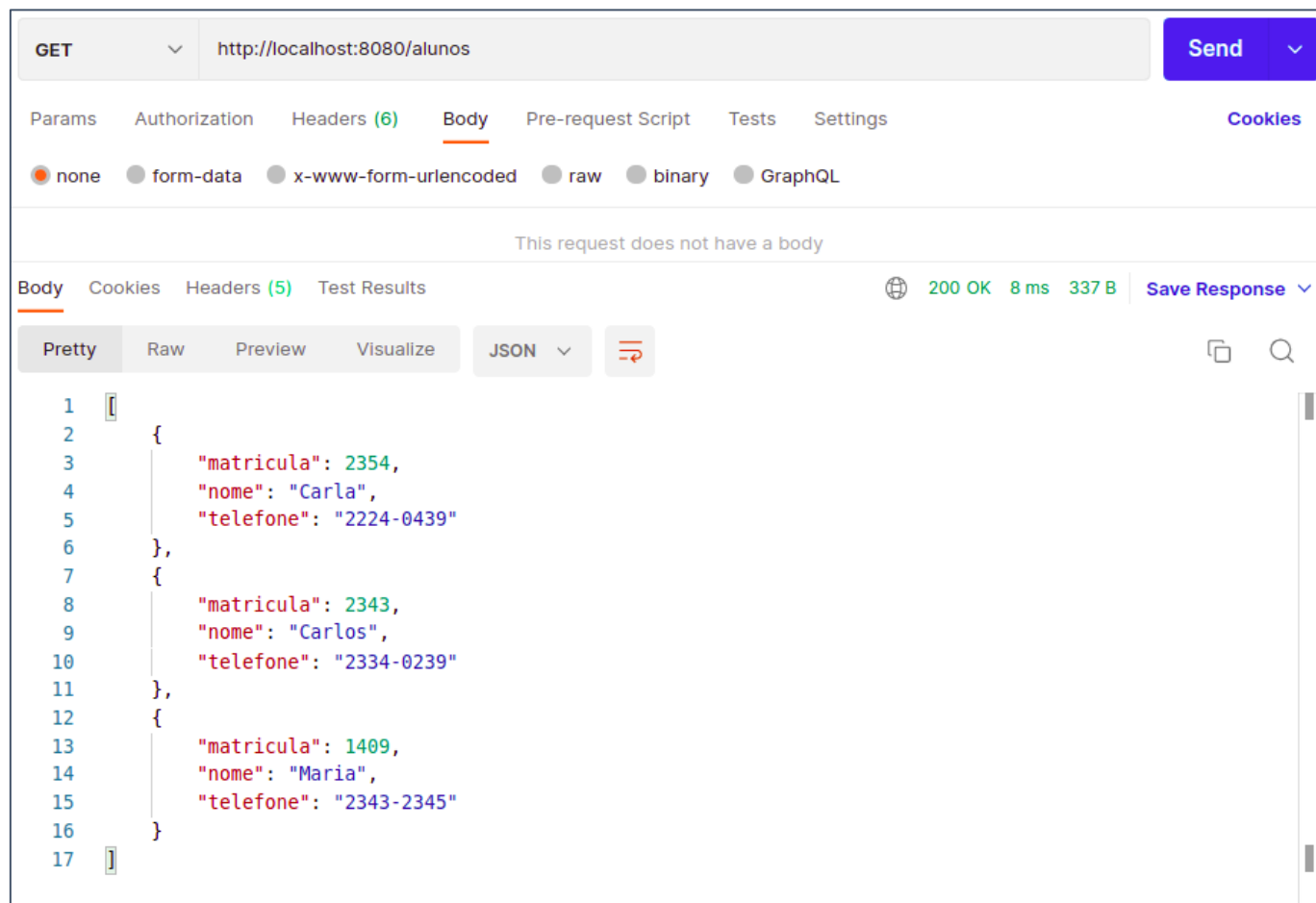
**@RequestMapping** - Indica que o recurso da API é /alunos, podemos acessar através da URL <http://localhost:8080/alunos>.

**@GetMapping** - Mapeamento do GET para o recurso **/alunos**

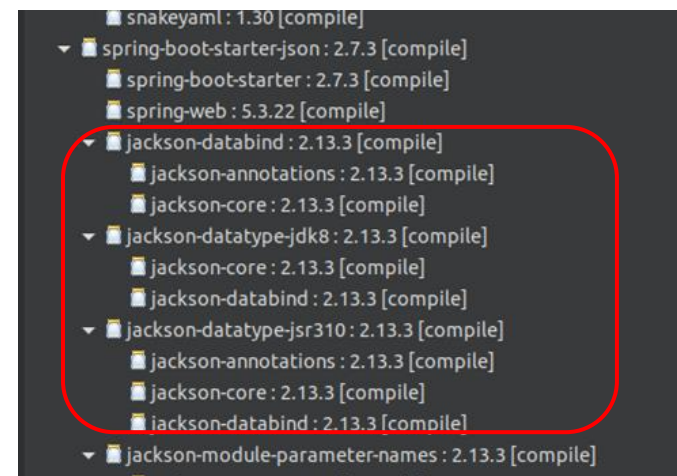
# Resultado



Fazendo o teste com o **Postman**



A biblioteca Jackson que vem como dependência no Spring Boot é responsável por converter a lista de objetos aluno em um json



# GetMapping e PathVariable



Criar o método para buscar uma matrícula

```
@GetMapping("/{matricula}")
public Aluno buscar(@PathVariable Long matricula) {
    for (int i = 0; i < lista.size(); i++) {
        if (lista.get(i).getMatricula().equals(matricula)) {
            return lista.get(i);
        }
    }
    return null;
}
```

**@PathVariable** - indica uma variável que faz parte do “path” da url que foi definido pela anotação **@RequestMapping**.

O código acima poderia ser substituído pela expressão lambda

```
return lista.stream().filter(a -> a.getMatricula().equals(matricula)).findFirst().orElse(null);
```

# GetMapping e PathVariable



No exemplo abaixo foi retornado o aluno de matrícula 2343.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/alunos/2343
- Buttons:** Send, Cookies
- Tabs:** Params, Authorization, Headers (9), Body (selected), Pre-request Script, Tests, Settings
- Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Body Tab:** Cookies, Headers (5), Test Results
- Response Status:** Status: 200 OK, Time: 13 ms, Size: 221 B, Save Response
- Response Format:** Pretty, Raw, Preview, Visualize, JSON (selected)
- Response Body (JSON):**

```
1 {  
2   "matricula": 2343,  
3   "nome": "Carlos",  
4   "telefone": "2334-0239"  
5 }
```

# PostMapping



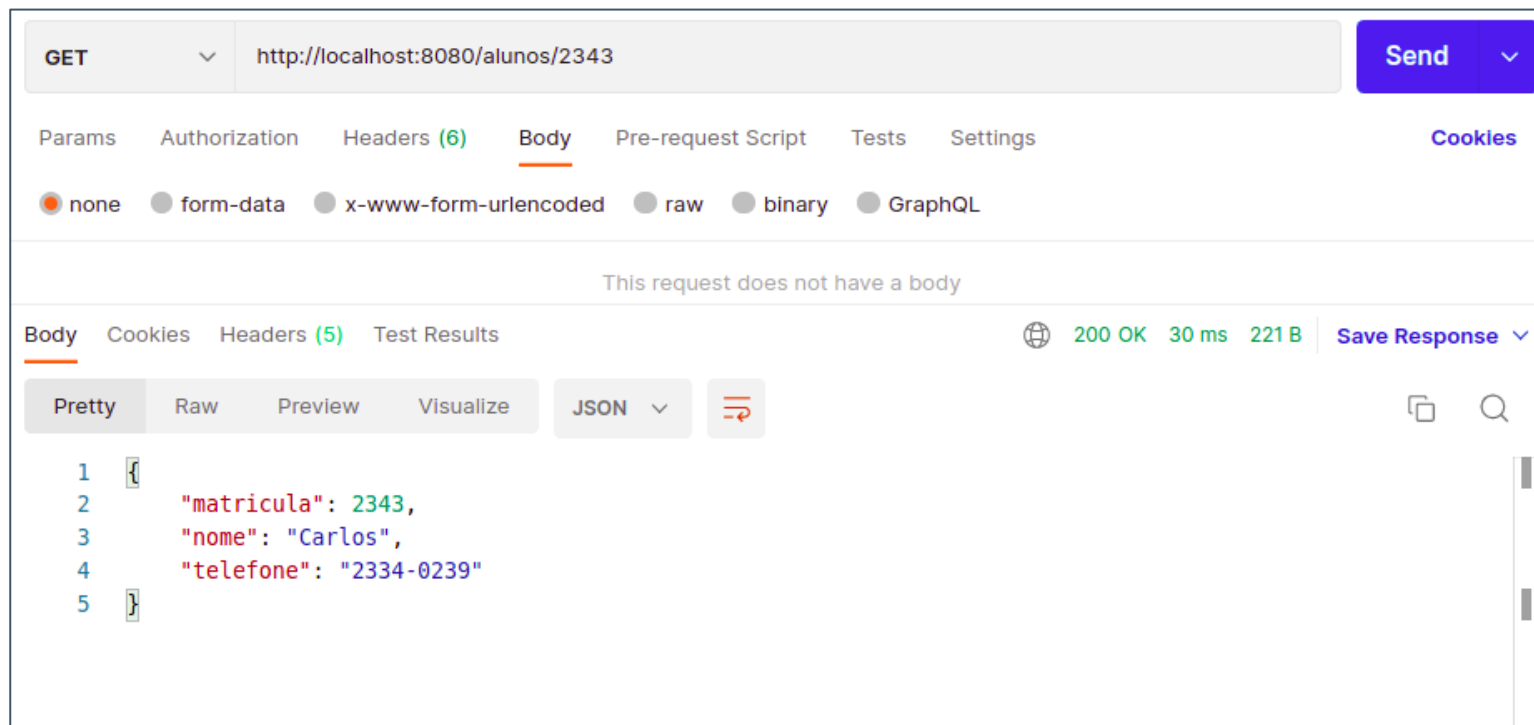
Criar o método para inserir um aluno

```
@PostMapping
public Aluno inserir(@RequestBody Aluno aluno) {
    lista.add(aluno);
    return aluno;
}
```

**@RequestBody** - Transformar o JSON que vem no corpo da requisição para um objeto Java. No exemplo o argumento **aluno** vai receber o conteúdo do JSON. A biblioteca jackson também responsável pela conversão das requisições, assim como ela faz nas respostas.



# PostMapping



Ao testarmos no Postman será retornado o código **200 OK** que está correto, mas temos um outro código específico para criação de novos registros que é o 201. No método podemos adicionar a anotação **@ResponseStatus(HttpStatus.CREATED)** conforme ao lado

```
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public Aluno inserir(@RequestBody Aluno aluno) {
    lista.add(aluno);
    return aluno;
}
```

# DeleteMapping



Criar o método para excluir um aluno incluindo a anotação @DeleteMapping

```
@DeleteMapping("/{matricula}")
public void delete(@PathVariable Long matricula) {
    for (int i = 0; i < lista.size(); i++) {
        if (lista.get(i).getMatricula().equals(matricula)) {
            lista.remove(i);
            break;
        }
    }
}
```

# DeleteMapping



Vamos fazer o teste com a exclusão do aluno com matrícula 2454

DELETE ▼ http://localhost:8080/alunos/2454 Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (4) Test Results 🌐 200 OK 8 ms 123 B Save Response ▼

Pretty Raw Preview Visualize Text ▼ ≡

1

# PutMapping



Criar o método para alterar os dados de um aluno incluindo a anotação @PutMapping

```
@PutMapping("/{matricula}")
public Aluno atualizar(@RequestBody Aluno aluno, @PathVariable Long matricula) {
    for (int i = 0; i < lista.size(); i++) {
        if (lista.get(i).getMatricula().equals(matricula)) {
            Aluno a = new Aluno(matricula, aluno.getNome(), aluno.getTelefone());
            lista.set(i, a);
            return a;
        }
    }
    return null;
}
```

# PutMapping



Vamos atualizar os dados do aluno com matrícula 2343.

The screenshot shows a REST client interface with a PUT request to `http://localhost:8080/alunos/2343`. The request body is a JSON object with the following fields:

```
1 {  
2   .... "matricula": 2343,  
3   .... "nome": "Mario Vitor",  
4   .... "telefone": "2247-7072"  
5 }
```

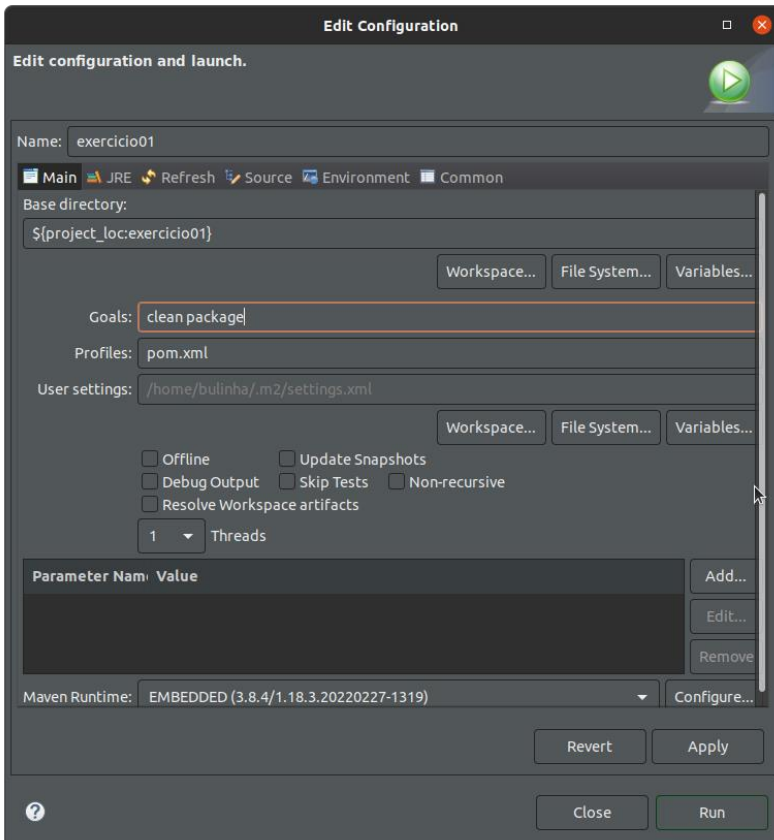
The response is also shown in the bottom panel, indicating a successful update with status `200 OK`, a response time of `59 ms`, and a body size of `226 B`. The response body is displayed in a pretty-printed JSON format:

```
1 {  
2   "matricula": 2343,  
3   "nome": "Mario Vitor",  
4   "telefone": "2247-7072"  
5 }
```

# Preparação do Deploy



Deploy, em inglês significa implantar. Em desenvolvimento de aplicações, significa colocar no ar a aplicação logo que esteja em um estado capaz de ser executada (sem erros de compilação pelo menos) e já “empacotada” (dependendo do tipo de aplicação).



Para fazermos o build da aplicação vamos utilizar o maven. Para gerar o arquivo **jar** clique com o botão direito no projeto

**Run as – Maven build – digite em Goals – clean package e clique em Run**

**clean** - limpa as dependências e gera novamente.

**package** - empacota o código de acordo com o tipo de extensão escolhido (jar ou war por exemplo).

Após clicar em Run o download das dependências é inicializado.

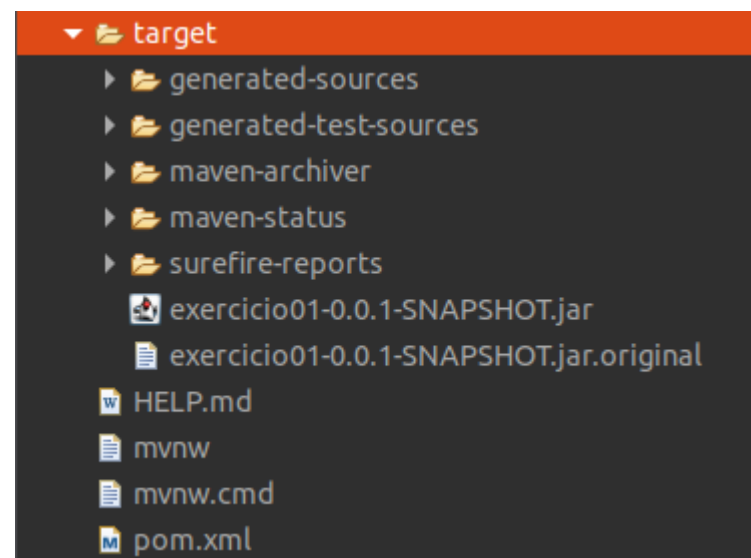
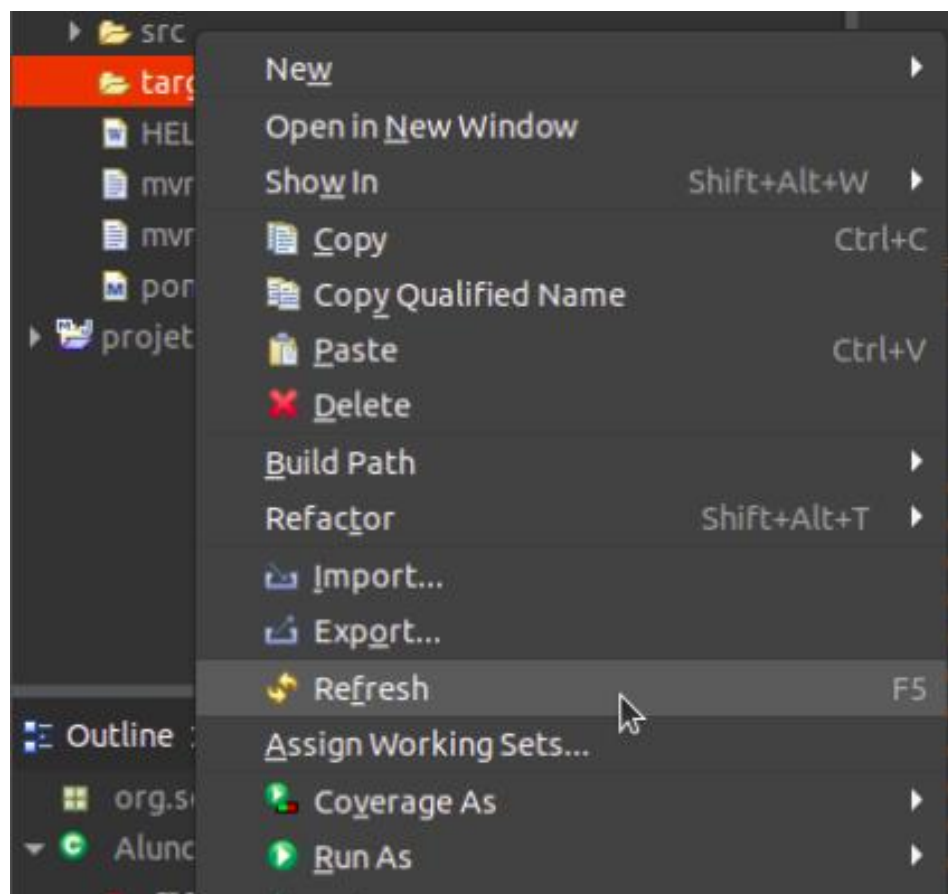
```
<terminated> exercicio01 [Maven Build] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (1 de set de 2022 17:12:05 – 17:13:00) [pid: 965294]
17:12:35.619 [main] DEBUG org.springframework.test.context.support.AbstractDirtyContextTestExecutionListener - Before test class: context [DefaultTest
Context]
=====
:: Spring Boot ::
(v2.7.3)

2022-09-01 17:12:36.405 INFO 965530 --- [main] o.s.b.e.Exercicio01ApplicationTests : Starting Exercicio01ApplicationTests using Java 1
2022-09-01 17:12:36.407 INFO 965530 --- [main] o.s.b.e.Exercicio01ApplicationTests : No active profile set, falling back to 1 default
2022-09-01 17:12:38.044 INFO 965530 --- [main] o.s.b.e.Exercicio01ApplicationTests : Started Exercicio01ApplicationTests in 2.391 seco
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.235 s - in org.serratec.backend.exercicio01.Exercicio01ApplicationTests
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ exercicio01 ---
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/3.5.2/maven-archiver-3.5.2.pom (5.5 kB at 3.8 kB/s)
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/3.5.2/maven-archiver-3.5.2.pom (5.5 kB at 3.8 kB/s)
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-artifact/3.1.1/maven-artifact-3.1.1.pom (2.0 kB at 6.3 kB/s)
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-artifact/3.1.1/maven-artifact-3.1.1.pom (2.0 kB at 6.3 kB/s)
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/maven/3.1.1/maven-3.1.1.pom (22 kB at 79 kB/s)
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/maven/3.1.1/maven-3.1.1.pom (22 kB at 79 kB/s)
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-model/3.1.1/maven-model-3.1.1.pom (22 kB at 79 kB/s)
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/maven-model/3.1.1/maven-model-3.1.1.pom (22 kB at 79 kB/s)
```

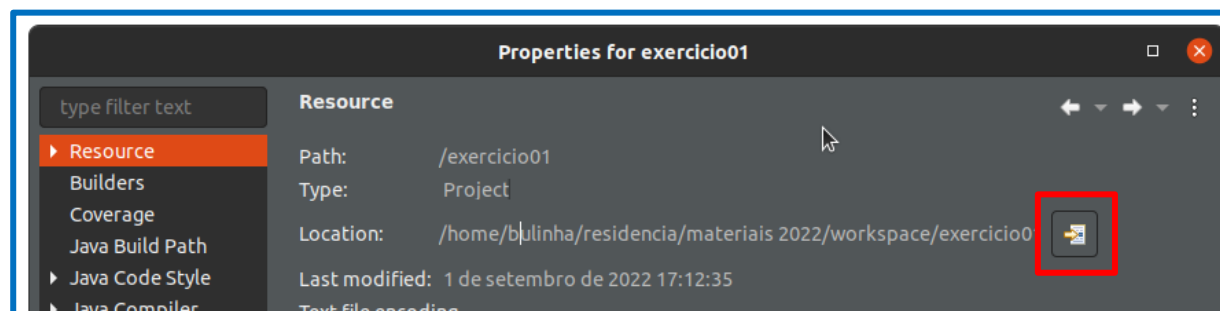
# Preparação do Deploy



Ao clicarmos com o botão direito na pasta **target** clique em **refresh** serão exibidas várias pastas temporárias e o arquivo .jar gerado pelo build.



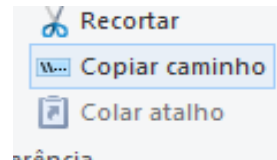
Vamos executar o arquivo gerado acessando para a pasta target em **Project** – **Properties** clique na seta em destaque, ao lado da localização do projeto



# Preparação do Deploy



Com o Windows Explorer aberto, selecione copiar caminho. Abrir o prompt de comando **Win+R** digite **cmd**



Digite **cd** e **CTRL+V** para colar o caminho. Digitar **java -jar** e o nome do arquivo

```
C:\WINDOWS\system32\cmd.exe

C:\Users\admin>cd "C:\Users\admin\Documents\workspace-spring-tool-suite-4-4.9.0.RELEASE\exercicio01\target"

C:\Users\admin\Documents\workspace-spring-tool-suite-4-4.9.0.RELEASE\exercicio01\target>java -jar exercicio01-0.0.1-SNAPSHOT.jar
```

```
C:\WINDOWS\system32\cmd.exe - java -jar exercicio01-0.0.1-SNAPSHOT.jar

Spring
=====
:: Spring Boot ::
(v2.4.2)

2021-02-14 17:44:03.454 INFO 15600 --- [main] o.s.b.e.Exercicio01Application : Starting Exercicio01Application v0.0.1-SNAPSHOT using Java 1.8.0_17
1 on LAPTOP-AFVR2PN2 with PID 15600 (C:\Users\admin\Documents\workspace-spring-tool-suite-4-4.9.0.RELEASE\exercicio01\target\exercicio01-0.0.1-SNAPSHOT.jar started by a
dmin in C:\Users\admin\Documents\workspace-spring-tool-suite-4-4.9.0.RELEASE\exercicio01\target)
2021-02-14 17:44:03.459 INFO 15600 --- [main] o.s.b.e.Exercicio01Application : No active profile set, falling back to default profiles: default
2021-02-14 17:44:06.904 INFO 15600 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-02-14 17:44:06.951 INFO 15600 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-02-14 17:44:06.951 INFO 15600 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2021-02-14 17:44:08.467 INFO 15600 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-02-14 17:44:08.468 INFO 15600 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4868 ms
2021-02-14 17:44:08.903 INFO 15600 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-02-14 17:44:09.380 INFO 15600 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-02-14 17:44:16.011 INFO 15600 --- [main] o.s.b.e.Exercicio01Application : Started Exercicio01Application in 13.896 seconds (JVM running for 1
```



# Exercício



- Criar uma classe com o nome Veículo com os atributos abaixo:
  - **id**
  - **marca**
  - **modelo**
- Criar uma classe com o nome **VeiculoController** e inserir as anotações para esta classe seja um controlador Rest.
- Criar uma lista de **Veiculos** e criar os principais métodos para adicionar, remover, listar, atualizar e buscar.
- Preparar o Deploy da API e executar.