

BEM-VINDOS!!

Residência de Software 2022/1

Disciplina: Desenvolvimento de API RestFull

- O que é uma API?
- JEE x Spring
- HTTP - Hyper Text Transfer Protocol
- Rest
 - POST - Inserir (**Create**) - INSERT
 - GET - Ler (**Read**) - SELECT
 - PUT - Atualizar (**Update**) - UPDATE
 - DELETE - Apagar (**Delete**) - DELETE

Apache Maven, ou Maven, é uma ferramenta de automação de compilação utilizada primariamente em projetos Java.

- Gerenciamento de Dependências
- Controle de Ciclo de vida
 - Validar projeto
 - Compilar
 - Testar - caso exista testes unitários
 - Empacotar - (criar o jar por exemplo)
 - Verificar - caso exista, testes de integração
 - Instalar - instalar o pacote num repositório local
 - Publicar (deploy) - instalar em repositório remoto
- Vasta opções de Plugins com inúmeras funções, ex.:
 - Execução de ferramentas externas
 - Geração de relatórios sobre o código
 - Validação de padrões de codificação (se a { do if vem após o último parênteses ou na próxima linha)



Anatomia de um pom

Project Object Model (POM)

- Identificadores de Projeto

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.baeldung</groupId>
  <artifactId>org.baeldung</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>org.baeldung</name>
  <url>http://maven.apache.org</url>
```

- Repositórios

```
<repositories>
  <repository>
    <id>JBoss repository</id>
    <url>http://repository.jboss.org/nexus/content/gr...
```

- Profiles (perfis)

```
<profiles>
  <profile>
    <id>production</id>
    <build>
      <plugins>
        <plugin>
          //...
        </plugin>
      </plugins>
    </build>
  </profile>
  <profile>
    <id>development</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <build>
      <plugins>
        <plugin>
          //...
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

- Propriedades

```
<properties>
  <spring.version>4.3.5.RELEASE</spring.version>
</properties>
```

- Dependências

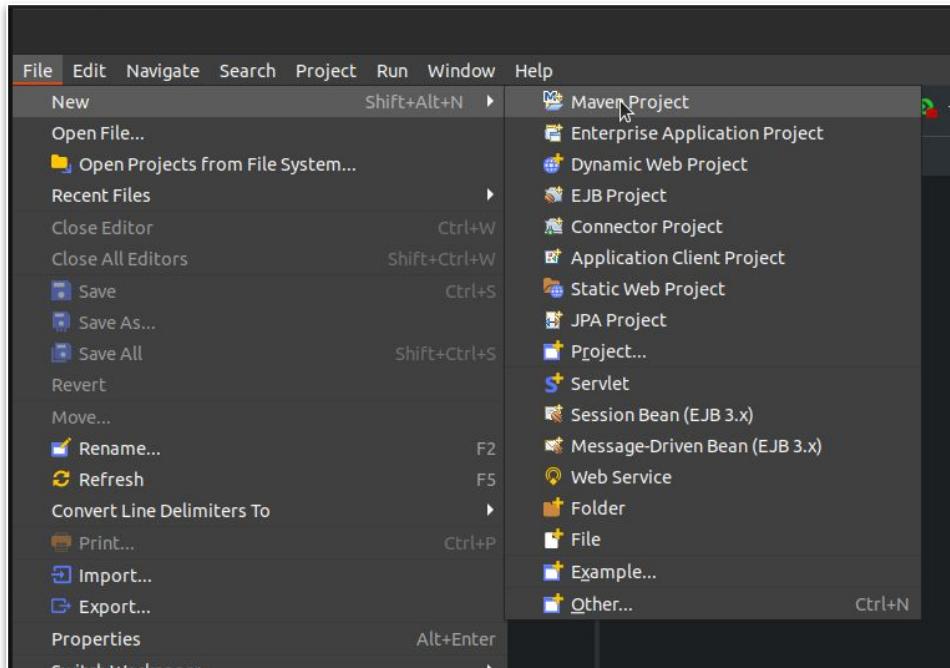
```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>
</dependencies>
```

- Build (construção)

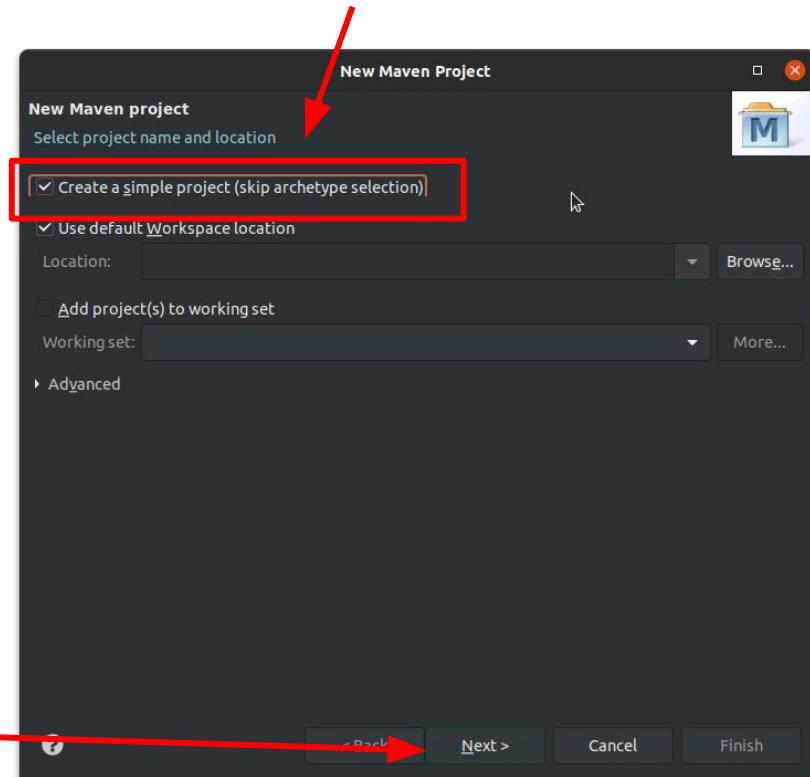
```
<build>
  <defaultGoal>install</defaultGoal>
  <directory>${basedir}/target</directory>
  <finalName>${artifactId}-${version}</finalName>
  <filters>
    <filter>filters/filter1.properties</filter>
  </filters>
  //...
</build>
```

Criando Projeto Maven

File > New > Maven Project



Selecionar a opção:
Create a simple project (skip archetype selection)

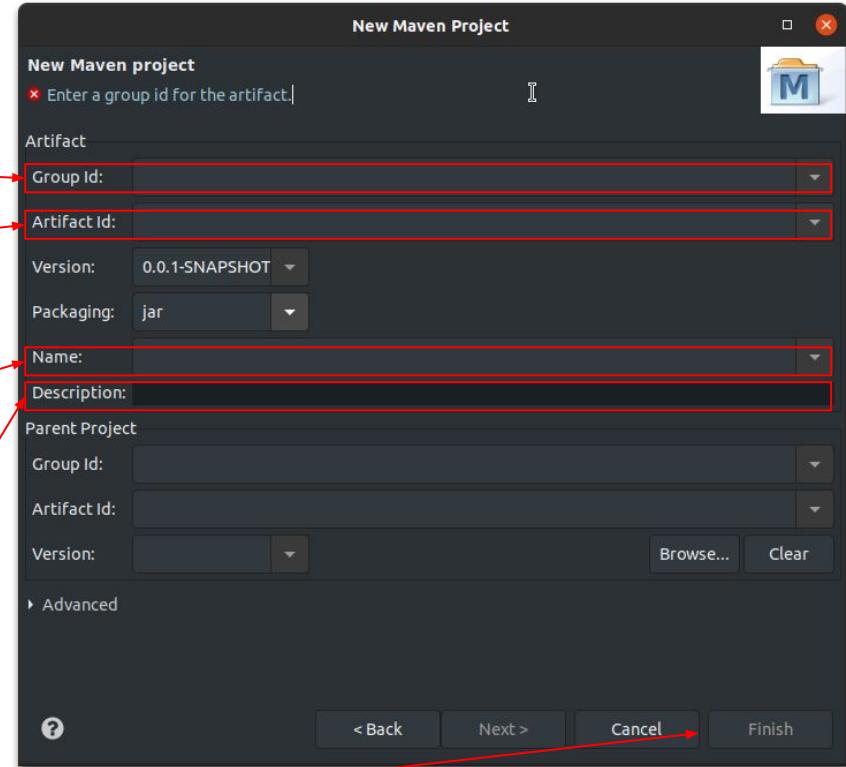


Clicar em Next

Criando Projeto 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

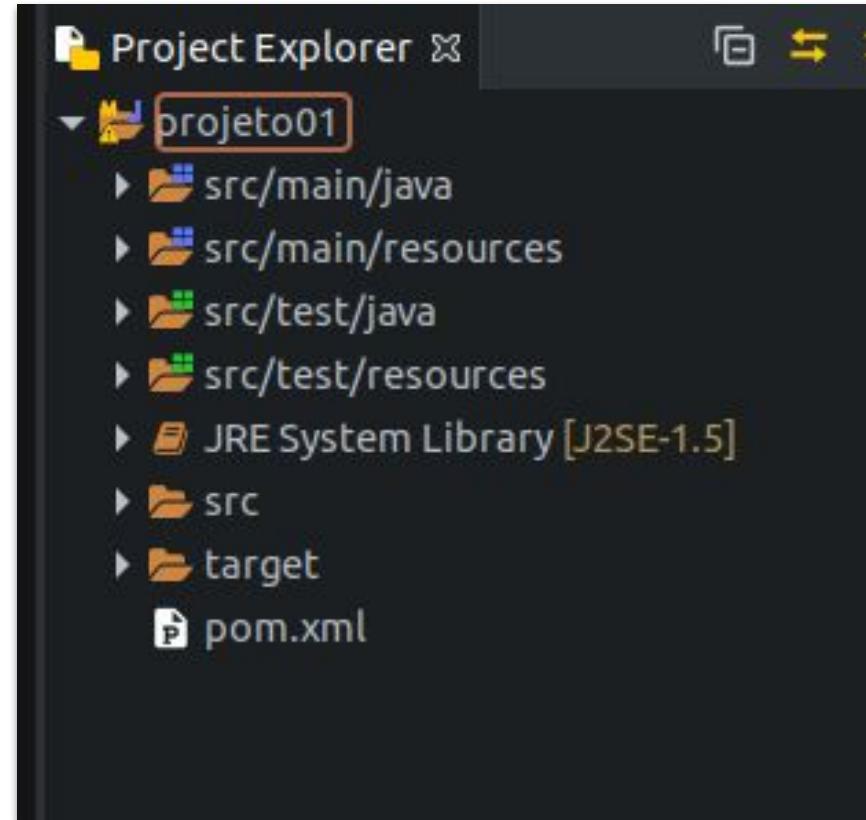


Clicar em Finish

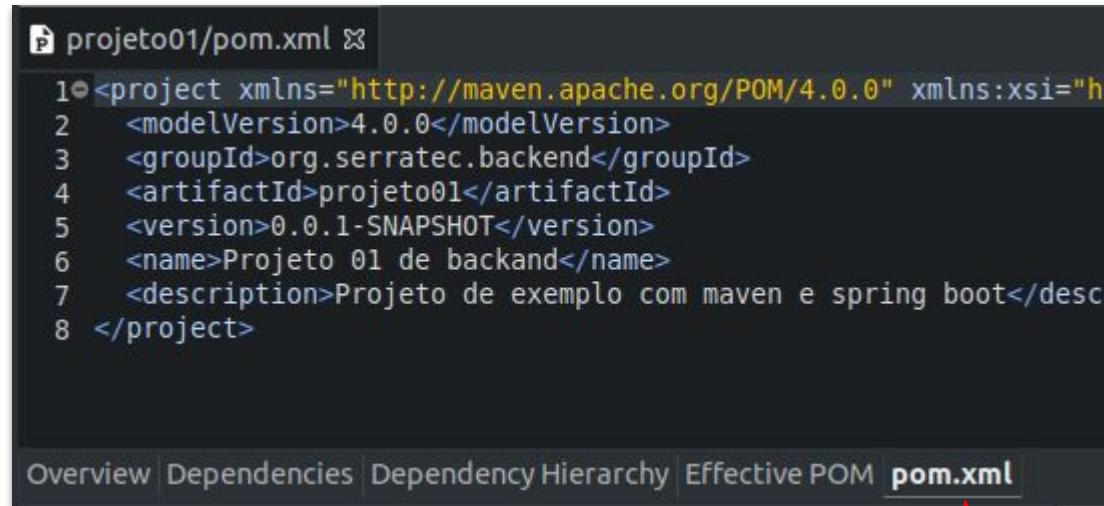
Criando Projeto Maven

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 com os dados que foi preenchido na tela de criação de projeto Maven



The screenshot shows a dark-themed interface for viewing Maven project files. At the top, there's a tab labeled 'projeto01/pom.xml'. Below it is the XML code for the pom.xml file:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
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</descr
8 </project>
```

At the bottom of the interface, there is a navigation bar with several tabs: 'Overview', 'Dependencies', 'Dependency Hierarchy', 'Effective POM', and 'pom.xml'. The 'pom.xml' tab is highlighted with a blue background and white text, and a red arrow points upwards from the bottom right towards this tab.

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

Convertendo p/Spring Boot

RESIDÊNCIA DE
SOFTWARE

serratec
Parque Tecnológico Região Serrana

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

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.2.RELEASE</version>
    <relativePath />
</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 backand</name>
    <description>Projeto de exemplo com maven e spring boot</de

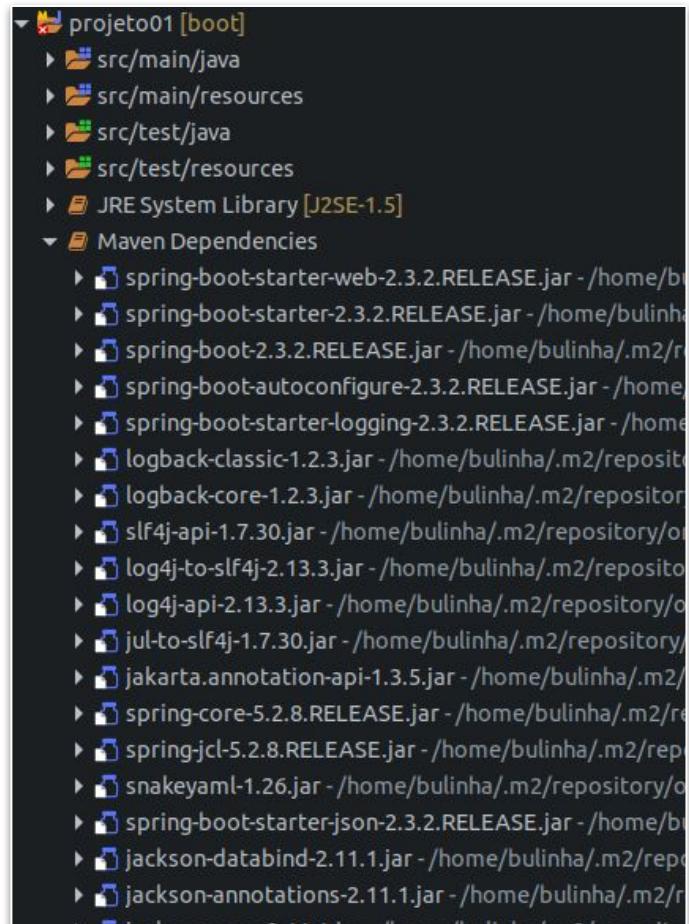
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.2.RELEASE</version>
    </parent>

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

</project>
```

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 dependencias hierarquicamente
- Cada um destes projetos tem seu próprio arquivo pom.xml



Convertendo p/Spring Boot

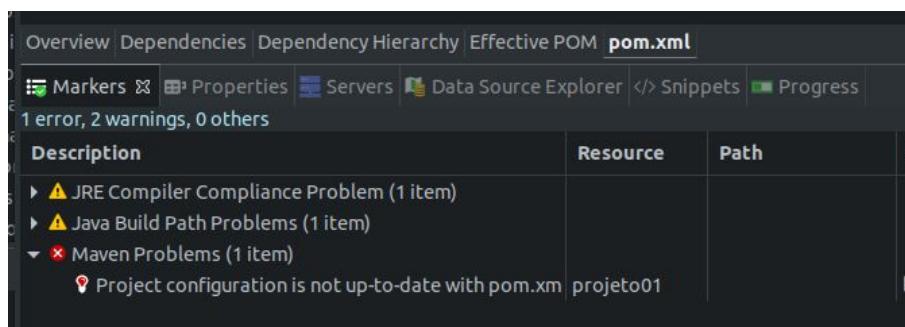
RESIDÊNCIA DE
SOFTWARE

serratec
Parque Tecnológico Região Serrana

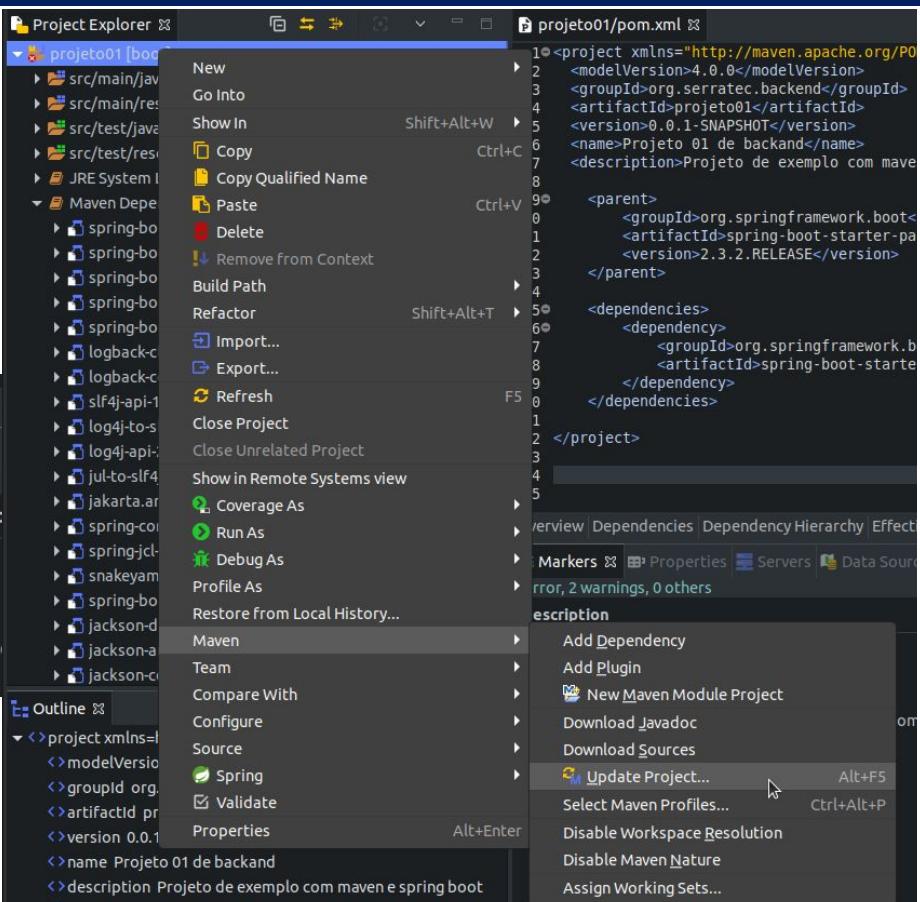
Ao salvar o projeto é possível perceber que há um erro sendo sinalizado no ícone no projeto



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

RESIDÊNCIA DE
SOFTWARE

serratec
Parque Tecnológico Região Serrana

Passo opcional: caso a versão do java sendo exibida na estrutura do projeto não seja a 1.8

```
<properties>
    <java.version>1.8</java.version>
</properties>
```



The screenshot shows a file explorer on the left with the project structure:

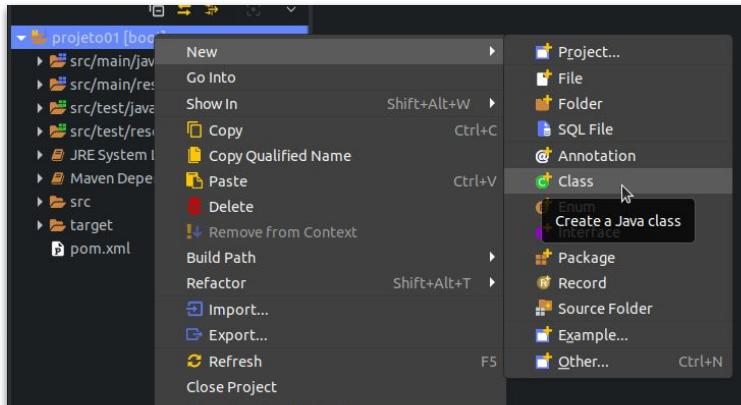
- projeto01 [boot]
- src/main/java
- src/main/resources
- src/test/java
- src/test/resources
 - JRE System Library [JavaSE-1.8] (highlighted with a red box)
- Maven Dependencies
- src
- target
- pom.xml

On the right is the content of the pom.xml file, with line numbers 1 through 19. Lines 9 through 12 are highlighted with a red box.

```
2   <modelVersion>1.0.0->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
9@   <properties>
10      <java.version>1.8</java.version>
11  </properties>
12
13@   <parent>
14      <groupId>org.springframework.boot</groupId>
15      <artifactId>spring-boot-starter-parent</artifactId>
16      <version>2.3.2.RELEASE</version>
17  </parent>
18
19@   <dependencies>
```

Criando Main

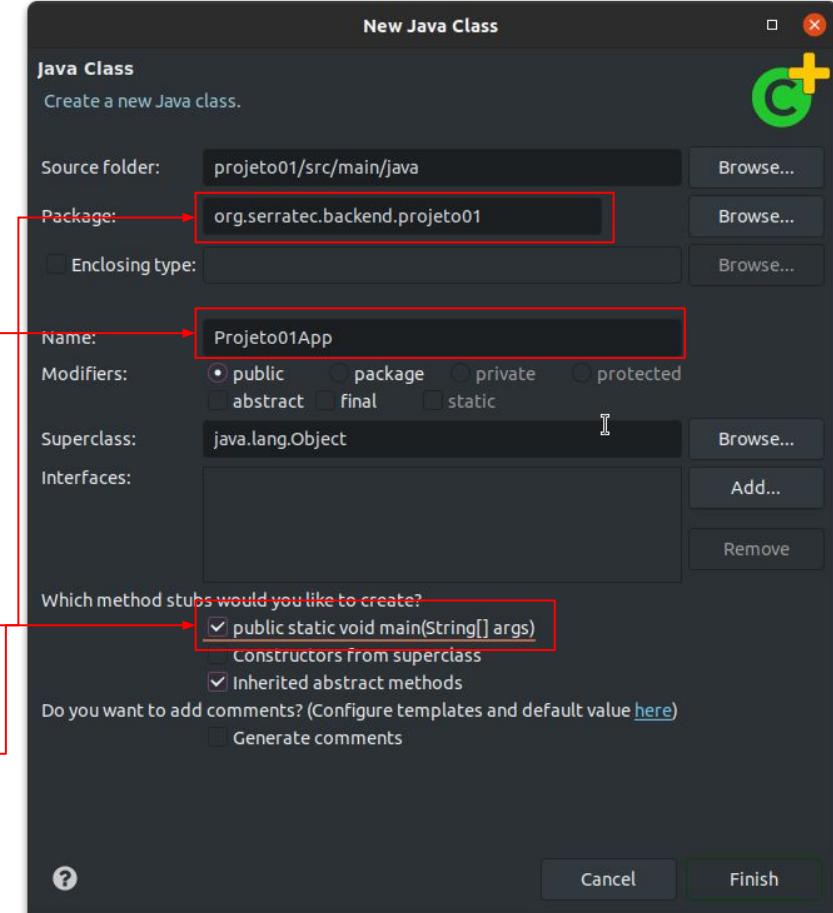
Clique com o botão direito no projeto, New > Class



Preencha os campo

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

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



Clique em Finish

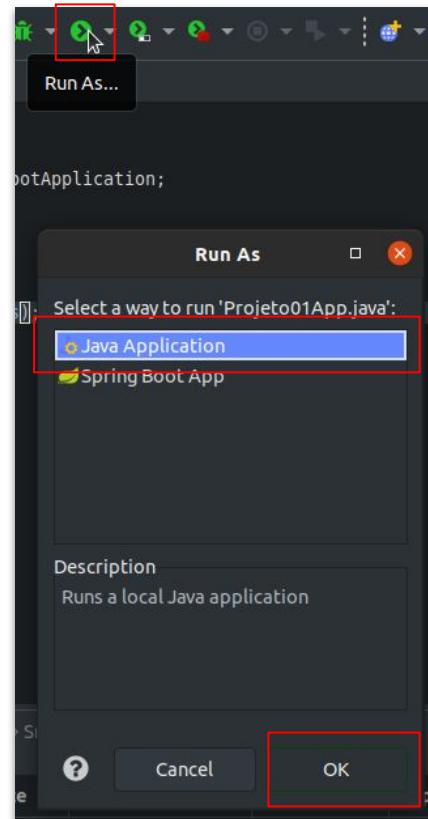
- `@SpringBootApplication`: indica a configuração padrão do SpringBoot
- `SpringApplication.run(Classe, args)`: Inicializa o SpringBoot e indica qual a classe inicial que ele irá utilizar e quais parâmetros ele receberá na inicialização

Feito isso, nossa aplicação Spring Boot está pronta para iniciar

```
J Projeto01App.java ✘
1 package org.serratec.backend.projeto01;
2
3 public class Projeto01App {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7     }
8
9 }
10
11 }
```

```
J Projeto01App.java ✘
1 package org.serratec.backend.projeto01;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Projeto01App {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Projeto01App.class, args);
11     }
12 }
13
14 }
```

Rodando



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

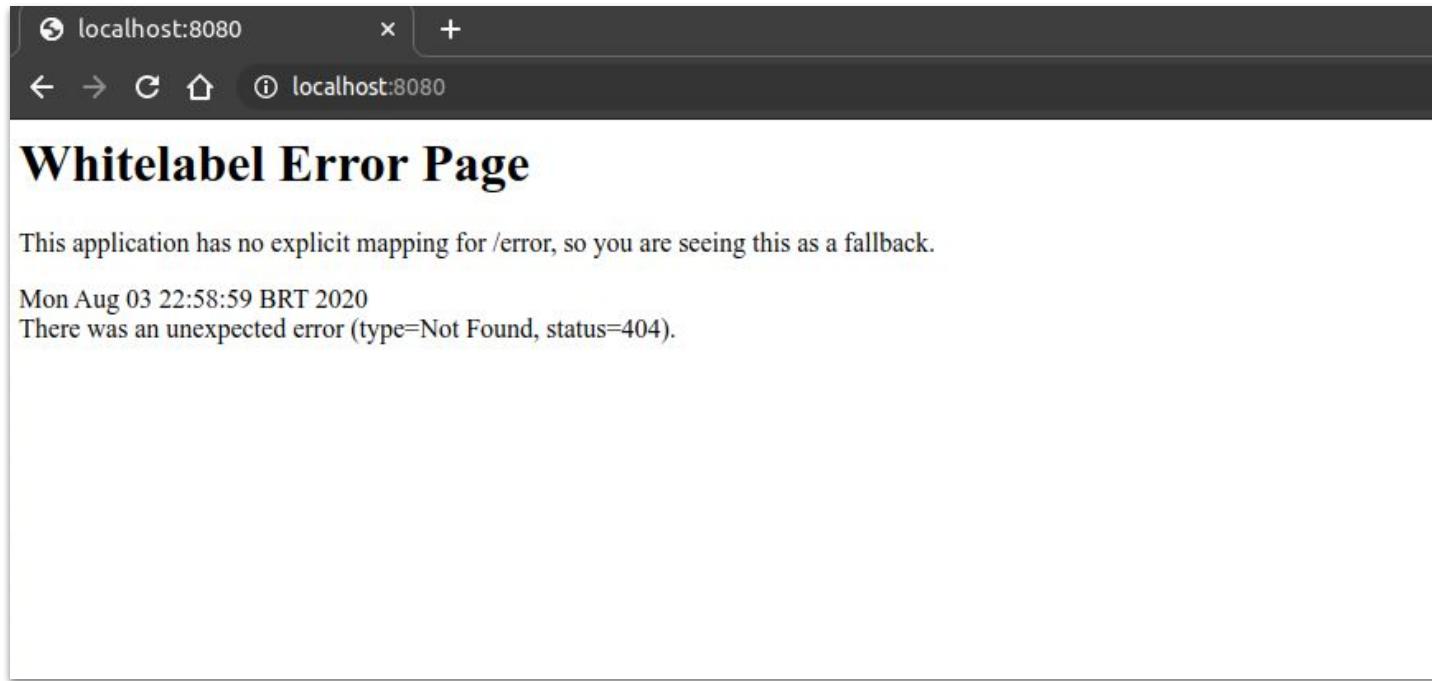
A janela do Console deve ganhar o foco automaticamente e será possível perceber ao final do “Log”, que a aplicação iniciou (Started) em 1.4 segundos (no caso do exemplo). E também é possível ver que ela usa o Tomcat “embeded” e que pode ser acessado pela porta 8080

A screenshot of the Eclipse IDE Console tab. The title bar shows 'Projeto01App [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (03/08/2020 22:53:15)'. The console window displays the Spring Boot logo and the message ': Spring Boot :: (v2.3.2.RELEASE)'. Below this, the log output shows the application starting up. A red box highlights the 'OK' button in the 'Run As' dialog from the previous screenshot. Another red box highlights the last line of the log output, which reads 'Started Projeto01App in 1.4 seconds (JVM running for 1.685)'.

```
2020-08-03 22:53:16.595 INFO 254318 --- [main] o.s.backend.projeto01.Projeto01App : Starting Projeto01App on bula-idx-notebook with PID 254318 (/home/bula-idx-notebook/.cache/eclipse/jee-2020-06/projeto01/Projeto01App)
2020-08-03 22:53:16.597 INFO 254318 --- [main] o.s.backend.projeto01.Projeto01App : No active profile set, falling back to default profiles: default
2020-08-03 22:53:17.319 INFO 254318 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-08-03 22:53:17.328 INFO 254318 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-08-03 22:53:17.329 INFO 254318 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2020-08-03 22:53:17.374 INFO 254318 --- [main] o.a.c.c.Tomcat : [localhost].[/]
2020-08-03 22:53:17.374 INFO 254318 --- [main] w.s.c.ServletWebServerApplicationContext : Initializing Spring embedded WebApplicationContext
2020-08-03 22:53:17.532 INFO 254318 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Root WebApplicationContext: initialization completed in 733 ms
2020-08-03 22:53:17.700 INFO 254318 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Initializing ExecutorService 'applicationTaskExecutor'
2020-08-03 22:53:17.712 INFO 254318 --- [main] o.s.backend.projeto01.Projeto01App : Tomcat started on port(s): 8080 (http) with context path ''
2020-08-03 22:53:17.712 INFO 254318 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Started Projeto01App in 1.4 seconds (JVM running for 1.685)
```

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

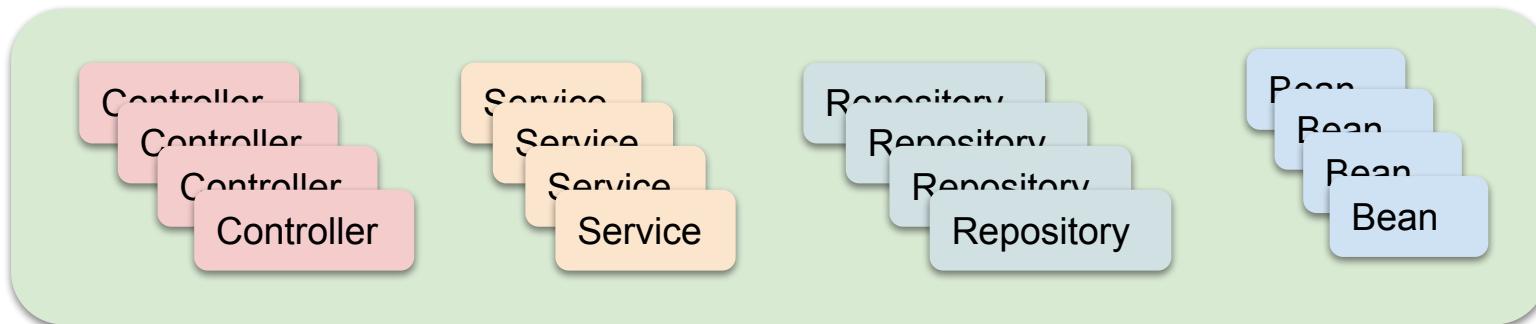
Como não temos nada desenvolvido e nem mesmo uma página de erro configurada, o Spring Boot apresenta esta página padrão.



```
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***

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

Via browser e/ou via Postman

```
package org.serratec.backend.projeto01;

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";
    }
}
```

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

Criando um Controller

The image shows a web browser window at the top displaying the URL `localhost:8080/ola`, which returns the response "Ola Mundo". Below this, a screenshot of the Postman application interface is shown, illustrating an API request.

Postman Request Screenshot:

- Method:** GET
- URL:** http://localhost:8080/ola
- Params:** (highlighted tab)
- Headers:** (9 items)
- Body:** (radio button selected)
- Pre-request Script:**
- Tests:**
- Settings:**

	KEY	VALUE	DES
	Key	Value	De

Body Tab Content:

- Body Type:** Text
- Content:** 1 Ola Mundo

Status: 200 OK

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

```
@RequestMapping("/maiusscula")
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")
```

The screenshot illustrates the interaction between a browser and a Postman API client. On the left, a browser window shows the URL `localhost:8080/maiusscula`. On the right, the Postman interface shows a GET request to `http://localhost:8080/maiusscula?valor=teste`. In the 'Params' tab of Postman, there is one entry: 'valor' with the value 'teste'. The 'Body' tab shows the response: 'TESTE'.

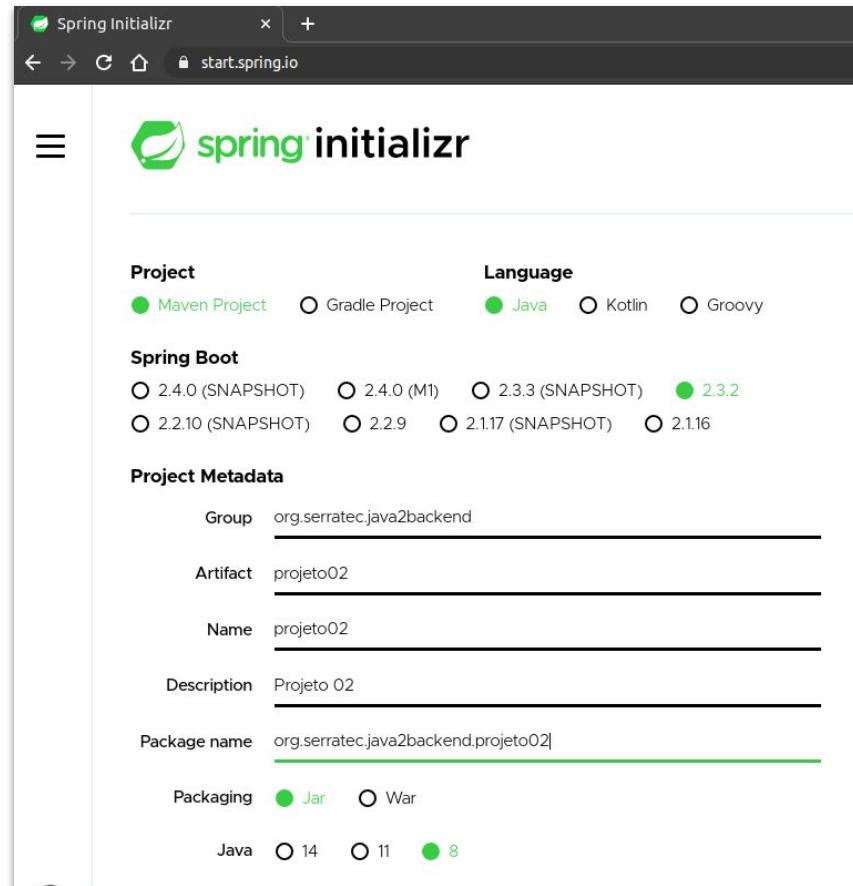
1. Implementar uma calculadora rudimentar
 - Somar
 - Multiplicar
 - Dividir
 - Subtrair

The screenshot shows the start.spring.io configuration interface. At the top, there's a browser header with the title "Spring Initializr" and the URL "start.spring.io". The main interface has several sections:

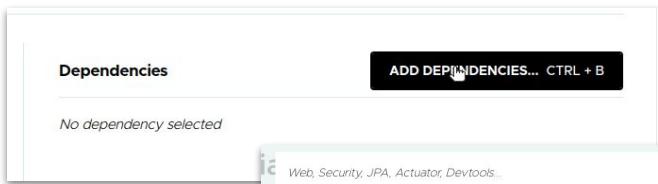
- Project**:
 - Maven Project
 - Gradle Project
- Language**:
 - Java
 - Kotlin
 - Groovy
- Dependencies**: A button labeled "ADD DEPENDENCIES... CTRL + B". Below it, the message "No dependency selected".
- Spring Boot**:
 - 2.4.0 (SNAPSHOT)
 - 2.4.0 (M1)
 - 2.3.3 (SNAPSHOT)
 - 2.3.2
 - 2.2.10 (SNAPSHOT)
 - 2.2.9
 - 2.1.17 (SNAPSHOT)
 - 2.1.16
- Project Metadata**:
 - Group: com.example
 - Artifact: demo
 - Name: demo
 - Description: Demo project for Spring Boot
 - Package name: com.example.demo
- Packaging**:
 - Jar
 - War
- Java**:
 - 14
 - 11
 - 8

Criar de maneira rápida um projeto SpringBoot

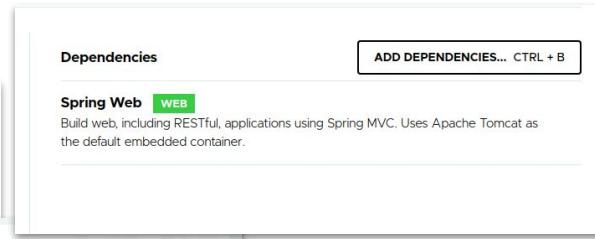
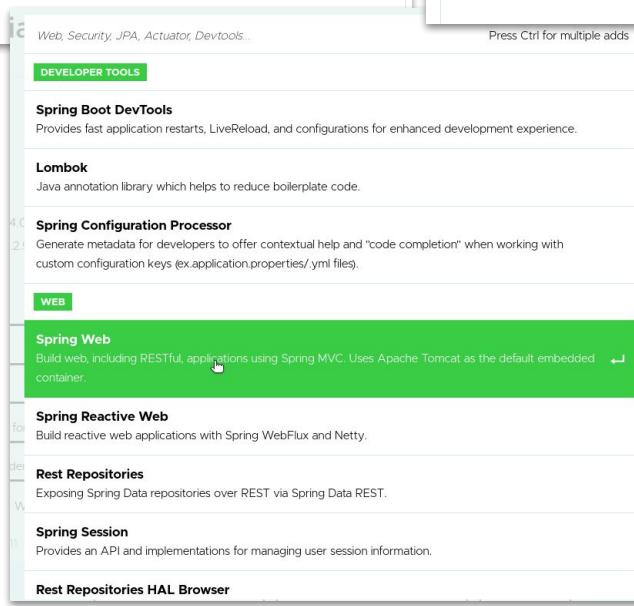
- Project: **Maven Project**
- Language: **Java**
- Spring Boot: **2.3.2**
- Project Metadata
 - Group: **org.serratec.java2backend**
 - Artifcat: **projeto02**
 - Name: **projeto02**
 - Description: **Projeto 02**
 - Package name:
org.serratec.java2backend.projeto02
 - Packaging: **Jar**
 - Java: **8**



1 - Adicionar dependências



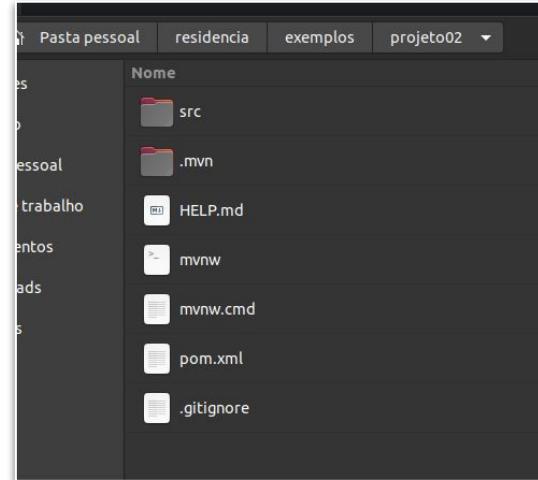
2 - Selecionar Spring Web



3 - Gerar o projeto (download)

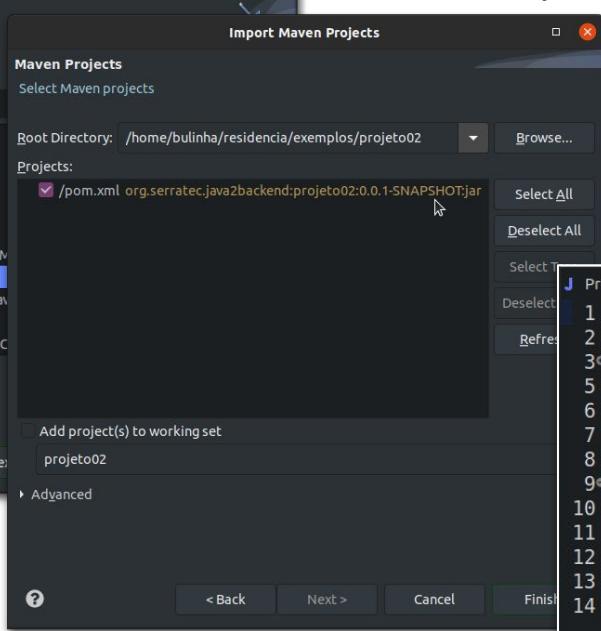
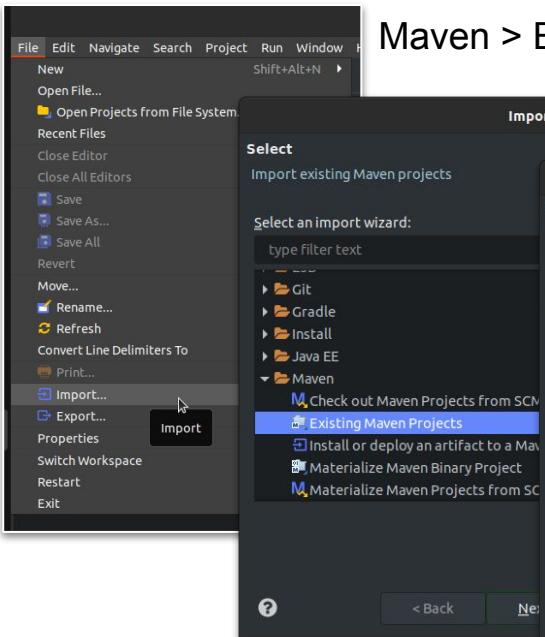


4 - Descompactar o zip



File > Import

Maven > Existing Maven Project



Selecionar a pasta descompactada

The screenshot shows the Eclipse IDE's 'Project Explorer' view on the right, displaying two projects: 'projeto01' and 'projeto02'. The 'projeto02' project is expanded, showing its directory structure including 'src/main/java', 'src/main/resources', 'src/test/java', and 'target'. In the center, a code editor window is open, showing the 'Projeto02Application.java' file with the following code:

```

1 package org.serratec.java2backend.projeto02;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class Projeto02Application {
7
8     public static void main(String[] args) {
9         SpringApplication.run(Projeto02Application.class, args);
10    }
11
12 }
13
14 
```

Controller - Objetos

Classe de Entidade/Domínio

1. Criar a classe ao lado
2. Botão Direito do Mouse > Source ou Alt + Shift + S
3. Generate Getters and Setters
4. Select All
5. Ok
6. Botão Direito do Mouse > Source ou Alt + Shift + S
7. Generate Constructor using Fields
8. Select All
9. Ok

Importante, o código será inserido na posição do cursor

```
package org.serratec.java2backend.projeto02.dominio;

public class Todo {
    private Integer id;
    private String titulo;
    private String descricao;
}
```

```
package org.serratec.java2backend.projeto02.dominio;

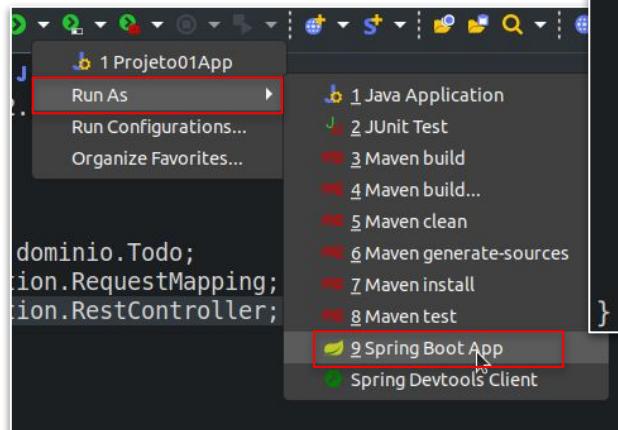
public class Todo {
    private Integer id;
    private String titulo;
    private String descricao;

    public Todo(Integer id, String titulo, String descricao) {
        super();
        this.id = id;
        this.titulo = titulo;
        this.descricao = descricao;
    }

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitulo() {
        return titulo;
    }
}
```

Controller - Objetos

1. Criar o Controle
2. Inicializar a aplicação
3. Acessar o a url com o Postman



```
package org.serratec.java2backend.projeto02.controllers;

import java.util.Arrays;
import java.util.List;

import org.serratec.java2backend.projeto02.dominio.Todo;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

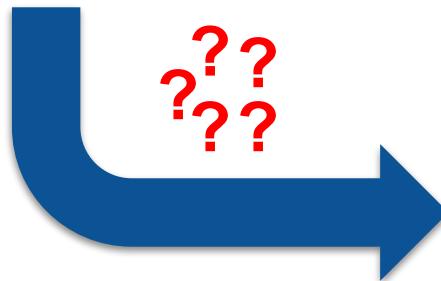
@RestController
public class TodoController {

    @RequestMapping("/todo")
    public List<Todo> getTodos(){
        return Arrays.asList(
            new Todo(1, "Compras", "Ir no mercado e fazer compras"),
            new Todo(2, "Remédio", "Passar na farmácia com a receita"),
            new Todo(3, "Aula", "Acessar o Teams para assistir a aula;")
        );
    }
}
```

Controller - Objetos

```
public class Todo {  
    private Integer id;  
    private String titulo;  
    private String descricao;  
  
    public Todo(Integer id, String titulo, String descricao)  
        super();  
        this.id = id;  
        this.titulo = titulo;  
        this.descricao = descricao;  
    }  
}
```

```
new Todo(1, "Compras", "Ir no mercado e fazer compras"),  
new Todo(2, "Remédio", "Passar na farmácia com a receita"),  
new Todo(3, "Aula", "Acessar o Teams para assistir a aula;")
```



GET http://localhost:8080/todo

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

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [ [  
2 {  
3     "id": 1,  
4     "titulo": "Compras",  
5     "descricao": "Ir no mercado e fazer compras"  
6 },  
7 {  
8     "id": 2,  
9     "titulo": "Remédio",  
10    "descricao": "Passar na farmácia com a receita"  
11 },  
12 {  
13     "id": 3,  
14     "titulo": "Aula",  
15     "descricao": "Acessar o Teams para assistir a aula;"  
16 }  
17 ] ]
```

Jackson - Obj -> Json

Biblioteca java para converter e manusear Json

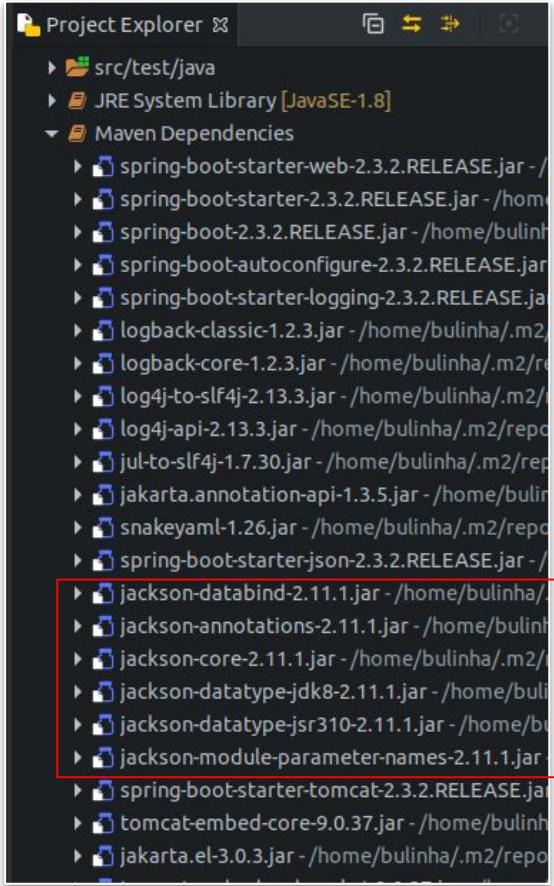
Gravar um objeto em um arquivo json

```
ObjectMapper objectMapper = new ObjectMapper();
Car car = new Car("yellow", "renault");
objectMapper.writeValue(new File("target/car.json"), car);

// conteúdo do arquivo
// {"color":"yellow", "type": "renault"}
```

Converter uma “string” json em objeto

```
String json = "{ \"color\" : \"Black\", \"type\" : \"BMW\" }";
Car carr = objectMapper.readValue(json, Car.class);
```



Mais informações <https://www.baeldung.com/jackson-object-mapper-tutorial>

Spring Boot - Sob o capô

“Herda” as configurações “padrão” do Spring

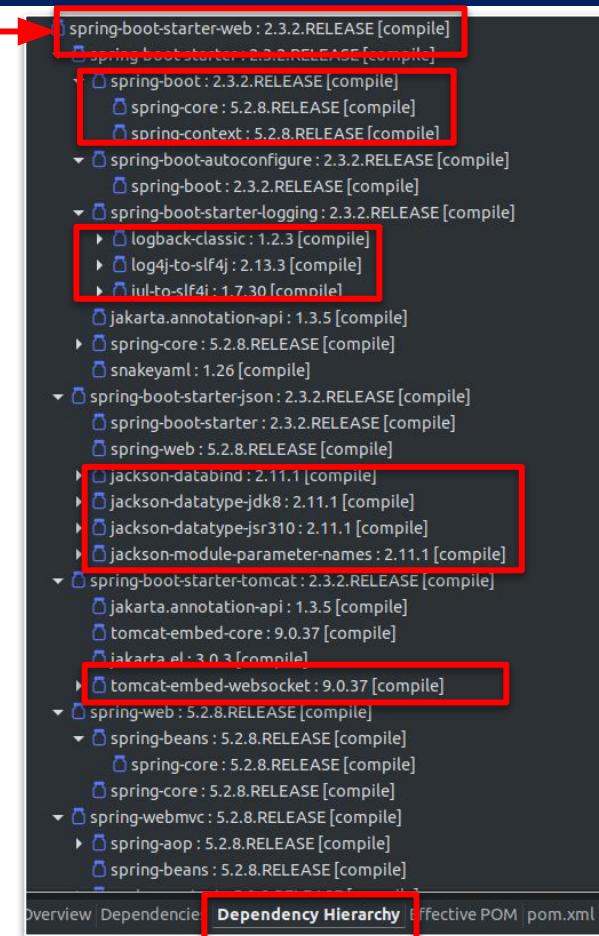
```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.2.RELEASE</version>
</parent>
```

Inclui todas as dependências necessárias para web e suas configurações padrão

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

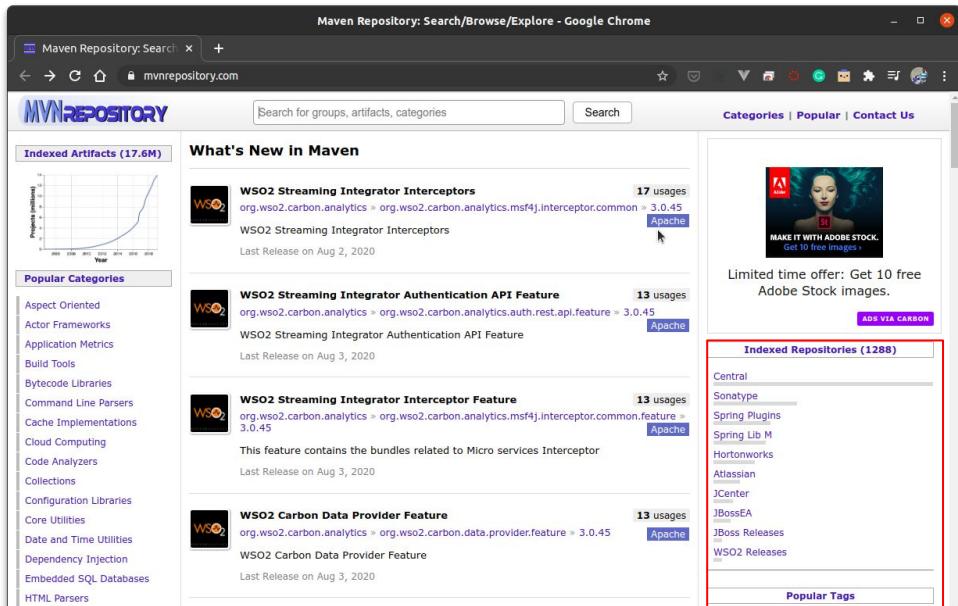
Bibliotecas de Log, Json, Spring-Core, Spring-Context e até o Tomcat “Embedded” (embutido)

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



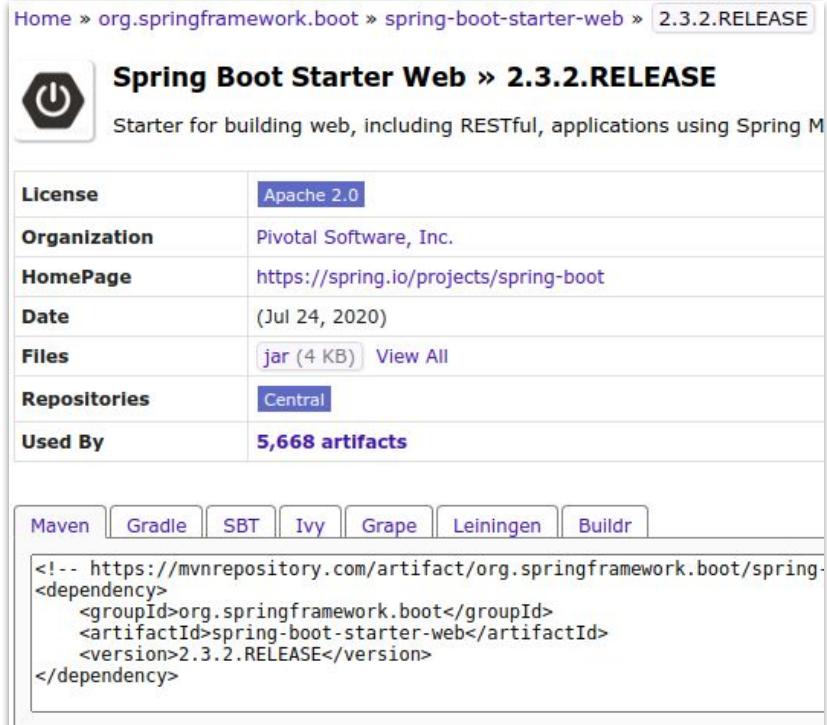
Repositórios Maven

Indexador de repositórios maven
<https://mvnrepository.com/>



The screenshot shows the Maven Repository search interface. On the left, there's a sidebar with 'Indexed Artifacts (17.6M)' and a graph showing the growth of indexed artifacts over time. Below it is a list of 'Popular Categories' including Actor Oriented, Application Metrics, and Build Tools. The main area displays 'What's New in Maven' with four recent releases from WSO2: 'WSO2 Streaming Integrator Interceptors', 'WSO2 Streaming Integrator Authentication API Feature', 'WSO2 Streaming Integrator Interceptor Feature', and 'WSO2 Carbon Data Provider Feature'. Each entry includes the group ID, artifact ID, version, number of usages, and the Apache license. A red arrow points to a section titled 'Indexed Repositories (1288)' which lists various Maven repositories like Central, Sonatype, Spring Plugins, and Atlassian. Another red arrow points to a 'Popular Tags' section at the bottom right of the page.

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



The screenshot shows the Maven Repository details page for the 'Spring Boot Starter Web' artifact. The URL is https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web/2.3.2.RELEASE. The page title is 'Spring Boot Starter Web > 2.3.2.RELEASE'. It provides information about the artifact, including its license (Apache 2.0), organization (Pivotal Software, Inc.), homepage (https://spring.io/projects/spring-boot), and release date (Jul 24, 2020). It also lists the file type (jar, 4 KB) and the number of artifacts used by other projects (5,668 artifacts). Below this, there are tabs for Maven, Gradle, SBT, Ivy, Grape, Leiningen, and Buildr. A code snippet shows the XML dependency declaration for this artifact:

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

Repositórios Maven

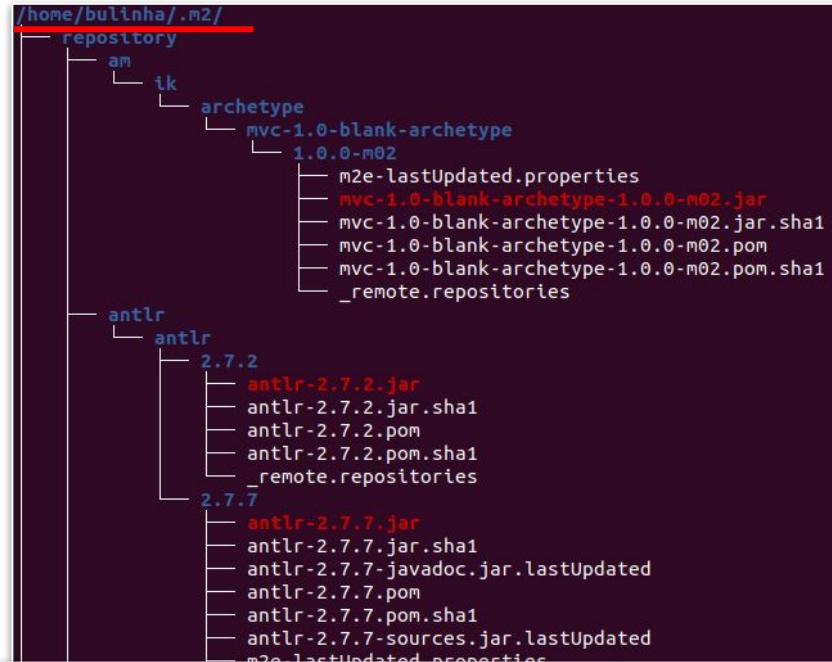
Repositório local: <home do usuário>/.m2

<home do usuário>

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

Bibliotecas são baixadas 1 vez e compartilhada entre todos os projetos maven na máquina

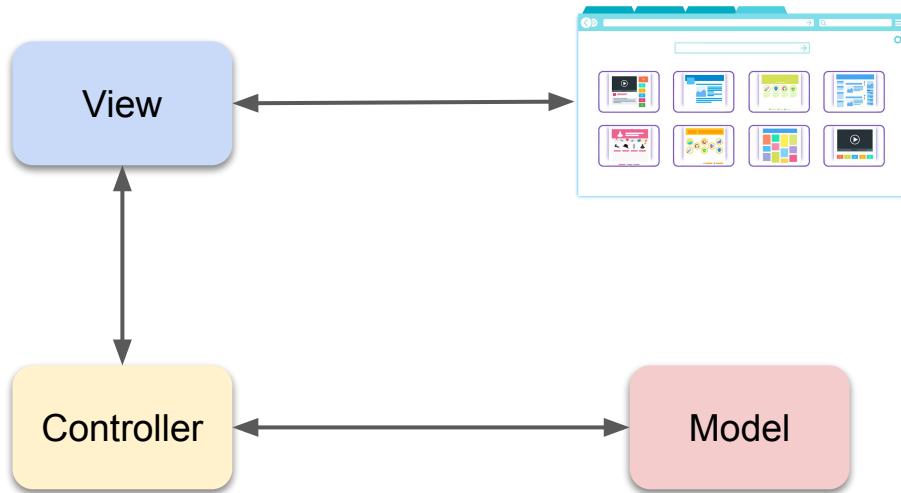
Estrutura de diretórios segue os valores do pom:
groupId, artifactId e número de versão



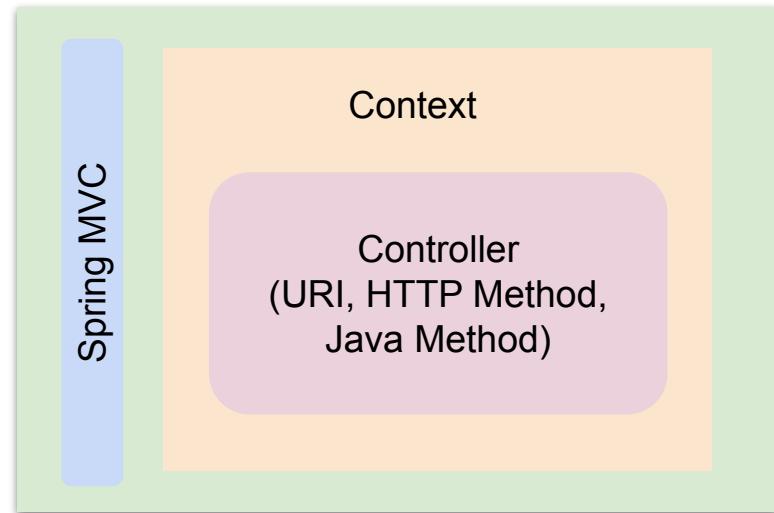
▼ Maven Dependencies

- ▶ spring-boot-starter-web-2.3.2.RELEASE.jar - /home/bulinha/.m2/repository/org/springframework/boot/spring-boot-starter-web/2.3.2.RELEASE
- ▶ spring-boot-starter-2.3.2.RELEASE.jar - /home/bulinha/.m2/repository/org/springframework/boot/spring-boot-starter/2.3.2.RELEASE
- ▶ spring-boot-2.3.2.RELEASE.jar - /home/bulinha/.m2/repository/org/springframework/boot/spring-boot/2.3.2.RELEASE
- ▶ spring-boot-autoconfigure-2.3.2.RELEASE.jar - /home/bulinha/.m2/repository/org/springframework/boot/spring-boot-autoconfigure/2.3.2.RELEASE
- ▶ spring-boot-starter-logging-2.3.2.RELEASE.jar - /home/bulinha/.m2/repository/org/springframework/boot/spring-boot-starter-logging/2.3.2.RELEASE

MVC - Model View Controller - Padrão de projeto



Spring MVC - implementação do padrão MVC



- Model

```
public class Todo {  
    private Integer id;  
    private String titulo;  
    private String descricao;  
  
    public Todo(Integer id, String titulo, String descricao)  
        super();  
        this.id = id;  
        this.titulo = titulo;  
        this.descricao = descricao;  
}
```

- Model - Classes Java - Entidades / Banco
- Controller - Recebe requisições, tratamento de erros HTTP, etc...
- View - Json, XML, Html, PDF, CSV, XLS, Imagen, etc....

- View

```
{  
    "id": 1,  
    "titulo": "Compras",  
    "descricao": "Ir no mercado e fazer compras"  
},  
{  
    "id": 2,  
    "titulo": "Remédio",  
    "descricao": "Passar na farmácia com a receita"  
},  
{  
    "id": 3,  
    "titulo": "Aula",  
    "descricao": "Acessar o Teams para assistir a aula;"  
}
```

- Controller

```
@RestController  
public class TodoController {  
  
    @RequestMapping("/todo")  
    public List<Todo> getTodos(){  
        return Arrays.asList(  
            new Todo(1, "Compras", "Ir no mercado e fazer compras"),  
            new Todo(2, "Remédio", "Passar na farmácia com a receita"),  
            new Todo(3, "Aula", "Acessar o Teams para assistir a aula;")  
        );  
    }  
}
```

Serviços são responsáveis pela regra de negócio de uma aplicação

Criar classe TodoService com a anotação @Service

```
package org.serratec.java2backend.projeto02.services;

import java.util.Arrays;
import java.util.List;

import org.serratec.java2backend.projeto02.dominio.Todo;
import org.springframework.stereotype.Service;

@Service
public class TodoService {

    private List<Todo> todos = Arrays.asList(
        new Todo(1, "Compras", "Ir no mercado e fazer compras"),
        new Todo(2, "Remédio", "Passar na farmácia com a receita"),
        new Todo(3, "Aula", "Acessar o Teams para assistir a aula")
    );

    public List<Todo> listaTodos(){
        return this.todos;
    }
}
```

Alterar o TodoController com um atributo privado que referencia a classe TodoService anotado com @Autowired

```
package org.serratec.java2backend.projeto02.controllers;

import java.util.List;

import org.serratec.java2backend.projeto02.dominio.Todo;
import org.serratec.java2backend.projeto02.services.TodoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class TodoController {
    @Autowired
    private TodoService todoService;

    @RequestMapping("/todo")
    public List<Todo> getTodos(){
        return todoService.listaTodos();
    }
}
```

IoC - Inversão de Controle

Nossa aplicação não é mais responsável por criar os componentes, o Spring passa a ser o responsável

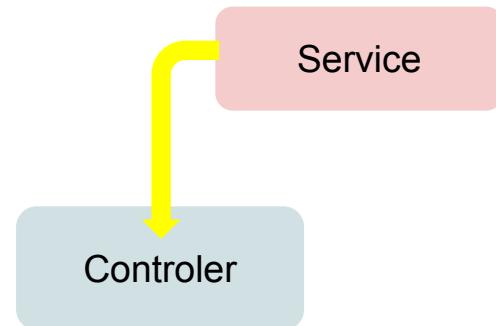
- Controllers
- Services
- Beans
- Etc..

~~new~~

Injeção de Dependência

Os componentes são “injetados” onde são necessários

- Atributos com anotação @Autowired
- Valores (componentes) requisitados no construtor de componentes



- **@SpringBootApplication** - utilizada pelo spring identificar o ponto de “partida” do seu projeto e “adivinhar algumas configurações, engloba outras 3 anotações: **@Configuration**, **@ComponentScan**, **@AutoConfiguration**
- **@EnableAutoConfiguration** - habilita a "auto-configuração" do Spring-Boot, ele irá procurar "Beans" no classpath e automaticamente colocá-los no contexto
- **@ComponentScan** - indica quais pacotes o Spring deve procurar por classes com anotações (por padrão ele considera o pacote da própria classe, mas pode receber um ou um array de pacotes)

```
@SpringBootApplication
class VehicleFactoryApplication {

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

```
@Configuration
@EnableAutoConfiguration
class VehicleFactoryConfig {}
```

```
@Configuration
@ComponentScan(basePackages = "com.baeldung.anno")
class VehicleFactoryConfig {}
```

- **@Autowired** - indica ao Spring que ele deve injetar um componente compatível com a classe/interface. Pode ser utilizado em variáveis de instância, métodos Set e Construtores (neste sendo opcional)

```
class Car {  
    @Autowired  
    Engine engine;  
}
```

```
class Car {  
    Engine engine;  
  
    @Autowired  
    void setEngine(Engine engine) {  
        this.engine = engine;  
    }  
}
```

```
class Car {  
    Engine engine;  
  
    @Autowired  
    Car(Engine engine) {  
        this.engine = engine;  
    }  
}
```

Spring Boot - Anotações (3)

- **@Configuration** - indica uma classe onde seus métodos podem ser utilizados para instanciar beans no contexto do Spring
- **@Bean** - indica um método responsável por instanciar objetos para o contexto do Spring, pode receber como parâmetro um “nome” para o bean

```
@Bean  
Engine engine() {  
    return new Engine();  
}
```

```
@Bean("engine")  
Engine getEngine() {  
    return new Engine();  
}
```

```
@Configuration  
class VehicleFactoryConfig {  
  
    @Bean  
    Engine engine() {  
        return new Engine();  
    }  
}
```

- **@Qualifier** - utilizado para indicar ao sistema de injeção de dependência do Spring, qual Bean deve ser utilizado, usando o nome que foi passado pela anotação Bean ou a classe específica da instância.

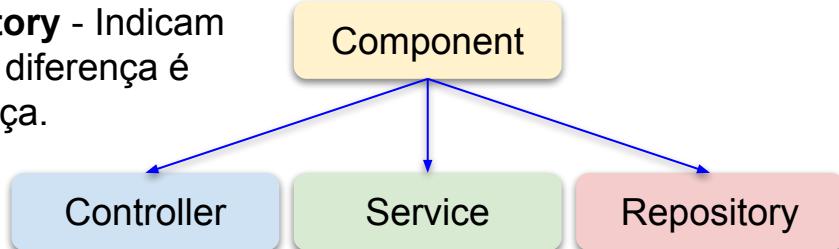
```
class Bike implements Vehicle {}  
  
class Car implements Vehicle {}
```

```
@Autowired  
@Qualifier("bike")  
Vehicle vehicle;
```

```
@Autowired  
void setVehicle(@Qualifier("bike") Vehicle vehicle) {  
    this.vehicle = vehicle;  
}
```

```
@Autowired  
Biker(@Qualifier("bike") Vehicle vehicle) {  
    this.vehicle = vehicle;  
}
```

- **@Component: @Controller, @Service e @Repository** - Indicam um Bean a ser armazenado no contexto do Spring. A diferença é puramente conceitual, funcionalmente não há diferença.



- **@RequestParam** - indica um parâmetro passado pela url deve ser passado para a variável parâmetro anotada no método. Pode ter seu “name” definido (caso seja diferente do parâmetro da url), bem como seu “defaultValue”.

```
@RequestMapping  
Vehicle getVehicleByParam(@RequestParam("id") long id) {  
    // ...  
}
```

```
@RequestMapping("/buy")  
Car buyCar(@RequestParam(defaultValue = "5") int seatCount) {  
    // ...  
}
```

- **@ResponseBody** - Indica que o retorno do método será utilizado como corpo da resposta HTTP. Se for um objeto (ou collection de objetos), usará o Jackson para converter em Json

```
@ResponseBody  
@RequestMapping("/hello")  
String hello() {  
    return "Hello World!";  
}
```

- **@RestController** - Combina a funcionalidade do @Controller com @ResposeBody para todos os métodos do controller. Pode receber como parametro um “path” que funcionará como tópico para todos os métodos mapeados do controller

```
@Controller  
@ResponseBody  
class VehicleRestController {  
    // ...  
}
```

```
@RestController  
class VehicleRestController {  
    // ...  
}
```

```
@RestController("/vehicle")  
class VehicleRestController {  
    // ...  
}
```

- **@RequestMapping** - indica o mapeamento de uma classe (tópico) e/ou de cada método da classe.
Possui diversos parâmetros
 - **path/name** - indica o path da url mapeada (se nenhum outro valor for passado, a string passada será considerada o path)
 - **method**: qual o método http que ele mapeia (RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE)
 - **consumes**: que tipo de dado ele “consoe” no corpo da requisição HTTP
 - **produces**: que tipo de dado ele “produz” no corpo da resposta HTTP
 - “application/json”
 - “application/xml”
 - MediaType.TEXT_PLAIN_VALUE

```
@RequestMapping("/home")
String home() {
    return "home";
}
```

```
@RequestMapping(value = "/vehicles/home", method = RequestMethod.GET)
String home() {
    return "home";
}
```

- **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping** - são “atalhos” para @RequestMapping com o parâmetro “method” já preenchido. Facilitam a leitura da classe.

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

```
@RequestMapping("/{id}")
Vehicle getVehicle(@PathVariable("id") long id) {
    // ...
}
```

```
@RequestMapping("/{id}")
Vehicle getVehicle(@PathVariable long id) {
    // ...
}
```

- **@RequestBody** - indica uma variável que irá receber o conteúdo do body do request HTTP

```
@PostMapping("/save")
void saveVehicle(@RequestBody Vehicle vehicle) {
    // ...
}
```

Criar um projeto que faça as operações de CRUD em uma lista de “Todo”:

- Utilizar um Service para armazenar a lista como variável de instância
 - Criar métodos para as operações
 - Create - Inserir um todo na lista
 - Read - Leitura, neste caso dois métodos: um que retorna todos os itens e um que retorna um único item com baseado no id passado como parâmetro para o método
 - Update - Atualizar, responsável por atualizar os valores dos atributos do Todo
 - Delete - Excluir, responsável por excluir um Todo da lista
 - Criar um controle que utilize o service criado acima e atenda a seguinte especificação de acesso rest:
 - GET /todo - retorna todos os “todos”
 - GET /todo/{n} - retorna o “todo” com id n
 - POST /todo - insere o “todo” que foi passado no corpo do request em json.
 - PUT /todo/{n} - atualiza os atributos do “todo” com id {n} utilizando os atributos passados no corpo do request em json
 - DELETE /todo/{n} - exclui o “todo” com o id {n} da lista