



# Desenvolvimento para Servidores-II

## Introdução ao Spring Boot

Neste tópico abordaremos o criação de um web service REST usando os recursos do Spring Boot

*Prof. Ciro Cirne Trindade*

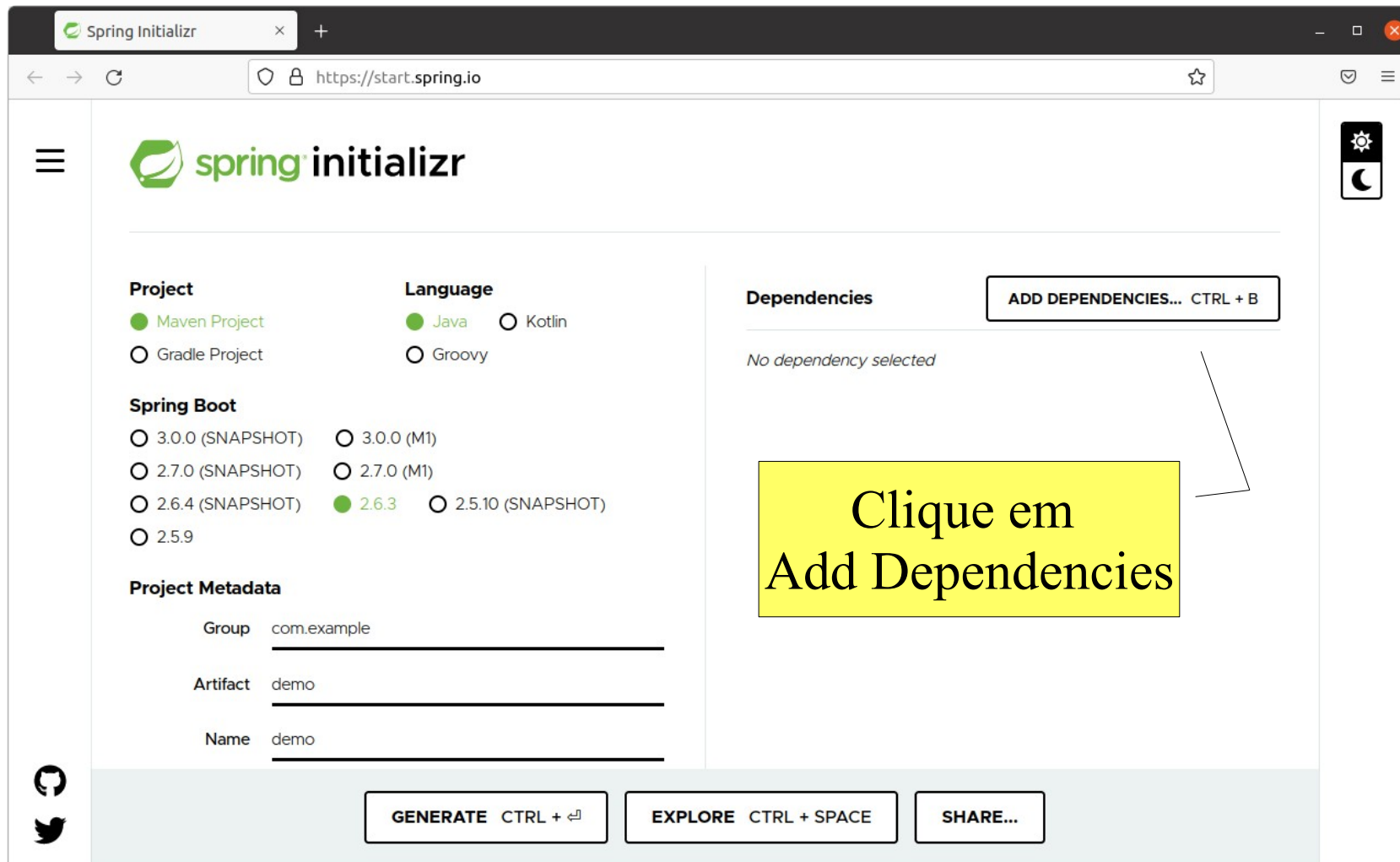
# Spring Boot

- <https://spring.io/projects/spring-boot>
- Spring Boot é um projeto da Spring IO que tem como principal objetivo alavancar a produtividade
  - Servidor Tomcat embutido
  - Simplifica a configuração das dependências
  - Configura automaticamente o Spring e bibliotecas de terceiros, quando possível
  - Nenhuma configuração em XML

# Spring Initializr (1/4)

- O Initializr oferece uma forma rápida de trazer todas as dependências que você precisa para uma aplicação e faz várias configurações para você
- <https://start.spring.io/>

# Spring Initializr (2/4)



The screenshot shows the Spring Initializr web application interface. The browser address bar displays `https://start.spring.io`. The page features a sidebar with a hamburger menu and a settings icon. The main content area is divided into sections for Project, Language, Spring Boot, Project Metadata, and Dependencies.

**Project**

- ☒ Maven Project
- ☐ Gradle Project

**Language**

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 3.0.0 (SNAPSHOT)
- ☐ 3.0.0 (M1)
- ☐ 2.7.0 (SNAPSHOT)
- ☐ 2.7.0 (M1)
- ☐ 2.6.4 (SNAPSHOT)
- ☒ 2.6.3
- ☐ 2.5.10 (SNAPSHOT)
- ☐ 2.5.9

**Project Metadata**

Group

Artifact

Name

**Dependencies**

No dependency selected

**Annotations:**

- A yellow box with the text "Clique em Add Dependencies" and an arrow pointing to the "ADD DEPENDENCIES... CTRL + B" button.

**Footer:**

- 
- 
-

# Spring Initializr (3/4)

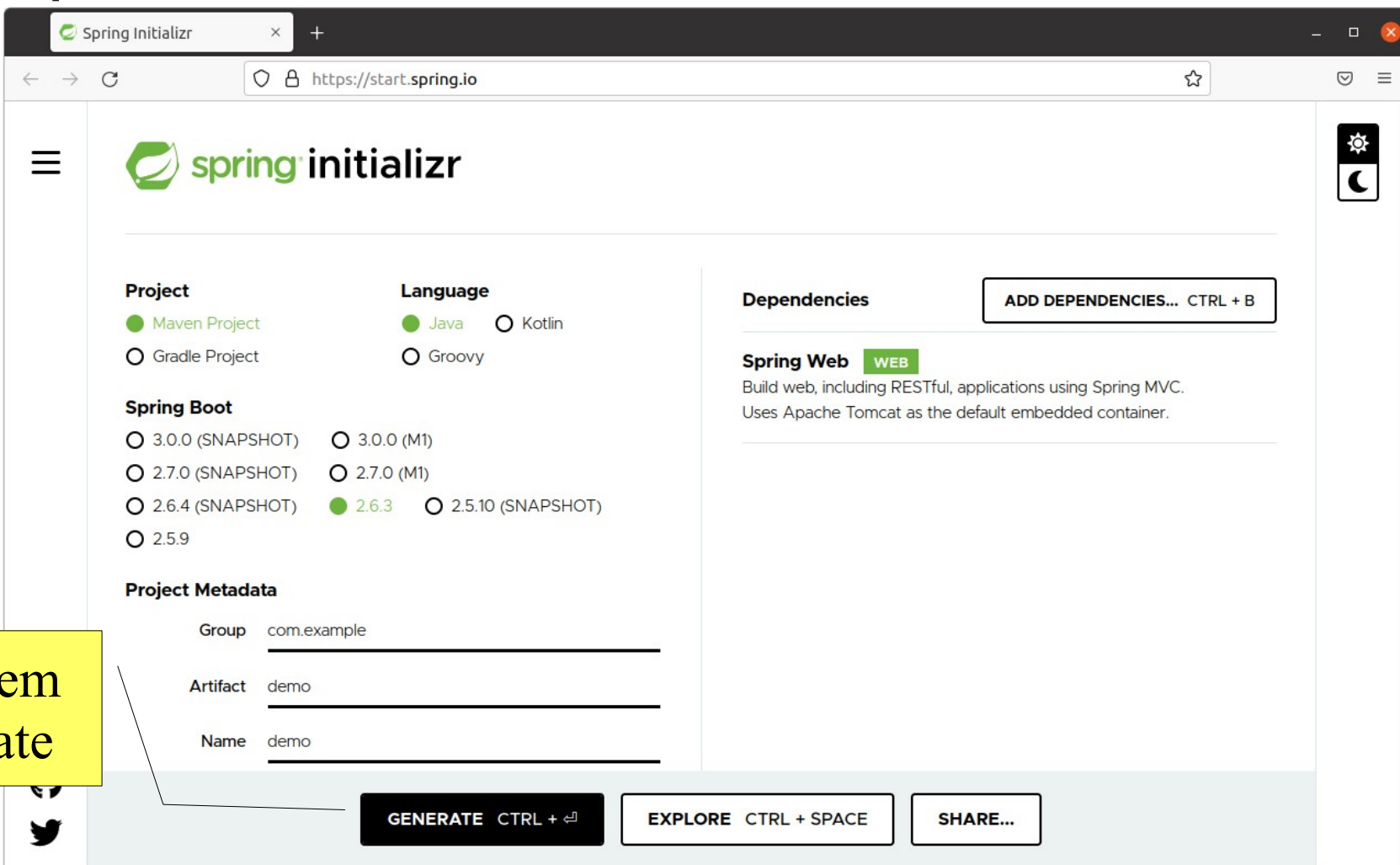
Digite web

Selecione  
Spring Web

web| Press Ctrl for multiple adds

<b>Spring Web</b>	WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.	
<b>Spring Reactive Web</b>	WEB
Build reactive web applications with Spring WebFlux and Netty.	
<b>Thymeleaf</b>	TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.	
<b>Spring Web Services</b>	WEB
Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.	
<b>WebSocket</b>	MESSAGING
Build WebSocket applications with SockJS and STOMP.	
<b>Jersey</b>	WEB

# Spring Initializr (4/4)



The screenshot shows the Spring Initializr web application in a browser window. The URL is <https://start.spring.io>. The interface includes a sidebar with a hamburger menu, the Spring logo, and a settings icon. The main content area is divided into three columns: Project, Language, and Dependencies. The Project column has radio buttons for Maven Project (selected), Gradle Project, and Spring Boot versions (3.0.0 (SNAPSHOT), 2.7.0 (SNAPSHOT), 2.6.4 (SNAPSHOT), 2.5.9, 3.0.0 (M1), 2.7.0 (M1), 2.6.3 (selected), and 2.5.10 (SNAPSHOT)). The Language column has radio buttons for Java (selected) and Kotlin, and Groovy. The Dependencies column has a button 'ADD DEPENDENCIES... CTRL + B' and a section for 'Spring Web' with a 'WEB' tag and a description. Below these columns is the 'Project Metadata' section with input fields for Group (com.example), Artifact (demo), and Name (demo). At the bottom, there are three buttons: 'GENERATE CTRL + G' (highlighted with a yellow box and a callout), 'EXPLORE CTRL + SPACE', and 'SHARE...'. A yellow box with the text 'Clique em Generate' and an arrow points to the 'GENERATE' button.

**Project**

- ☒ Maven Project
- ☐ Gradle Project

**Language**

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 3.0.0 (SNAPSHOT)
- ☐ 2.7.0 (SNAPSHOT)
- ☐ 2.6.4 (SNAPSHOT)
- ☐ 2.5.9
- ☐ 3.0.0 (M1)
- ☐ 2.7.0 (M1)
- ☒ 2.6.3
- ☐ 2.5.10 (SNAPSHOT)

**Project Metadata**

Group

Artifact

Name

**Dependencies** [ADD DEPENDENCIES... CTRL + B](#)

**Spring Web** **WEB**

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

**GENERATE** CTRL + G

**EXPLORE** CTRL + SPACE

**SHARE...**

Clique em  
Generate

# Executando a aplicação no Eclipse (1/6)

- Baixe e descompacte o arquivo
- Importe o projeto no Eclipse
  - File → Import → Maven → Existing Maven Project
  - Selecione a pasta do projeto e clique no botão Finish

# Executando a aplicação no Eclipse (2/6)

- Você verá apenas duas dependências configuradas o arquivo pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Isso corresponde a  
+ de 60 arquivos .jar!





# Executando a aplicação no Eclipse (3/6)

- O Initializr cria uma classe simples, anotada com **@SpringBootApplication**
  - Define a classe com a classe principal da aplicação
- Não há nenhuma configuração via XML

```
package com.example.demo;
```

```
import org.springframework.boot.SpringApplication;
```

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

```
@SpringBootApplication
```

```
public class DemoApplication {
```

```
    public static void main(String[] args) {
```

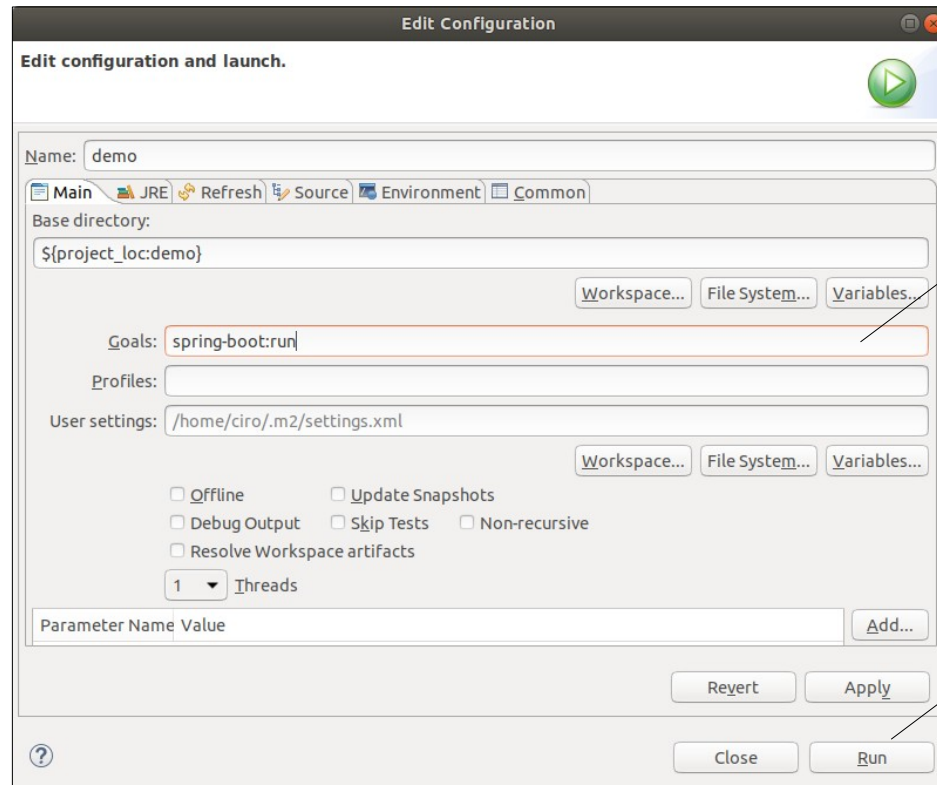
```
        SpringApplication.run(DemoApplication.class, args);
```

```
    }
```

```
}
```

# Executando a aplicação no Eclipse (4/6)

- Para executar a aplicação, clique com o botão direito do mouse sobre o projeto, selecione Run as → Maven Build

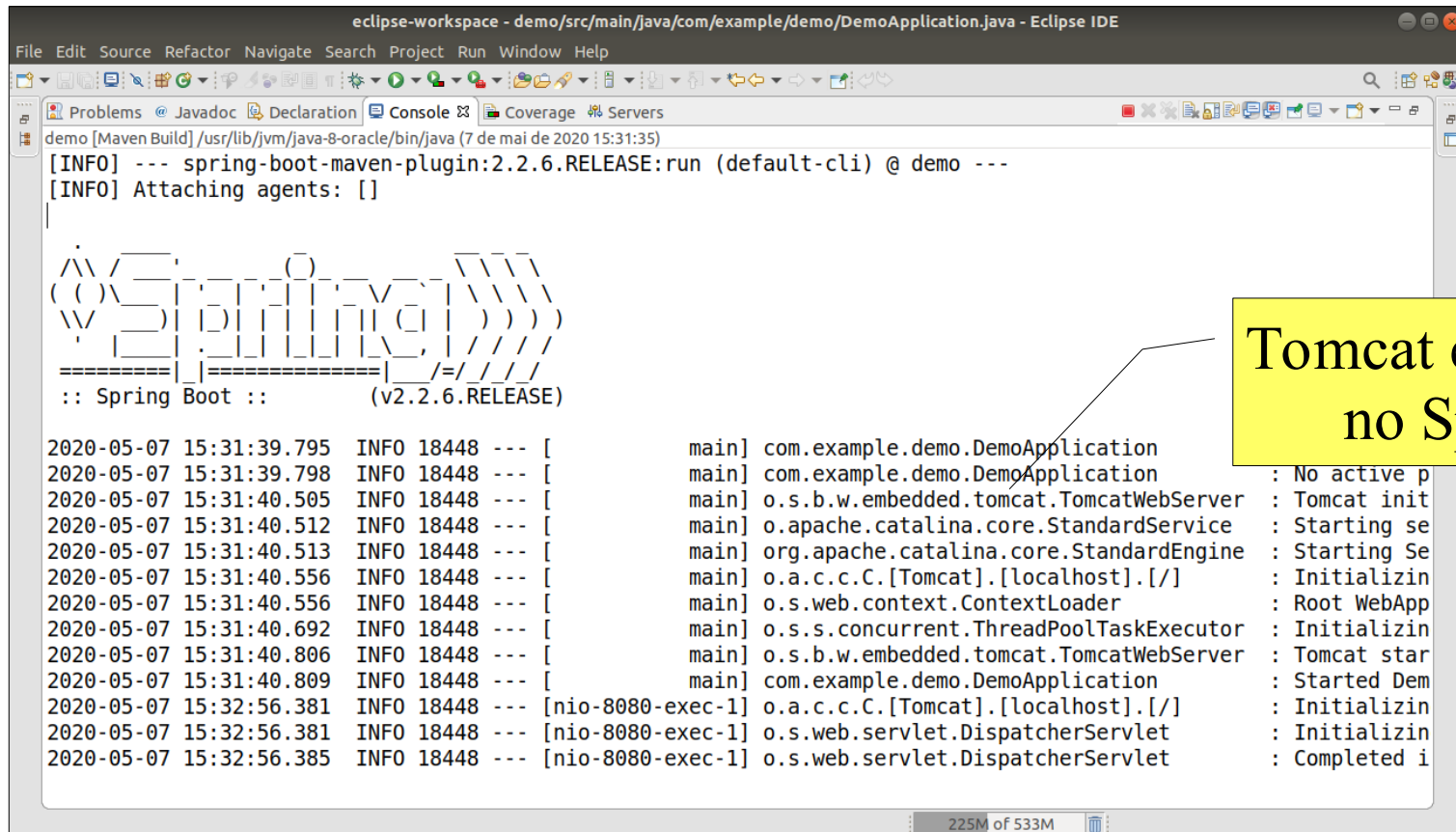


Goals: spring-boot:run

Clique em Run

# Executando a aplicação no Eclipse (5/6)

- A aplicação irá ser implantada num servidor Tomcat embutido



```
eclipse-workspace - demo/src/main/java/com/example/demo/DemoApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
demo [Maven Build] /usr/lib/jvm/java-8-oracle/bin/java (7 de mai de 2020 15:31:35)
[INFO] --- spring-boot-maven-plugin:2.2.6.RELEASE:run (default-cli) @ demo ---
[INFO] Attaching agents: []

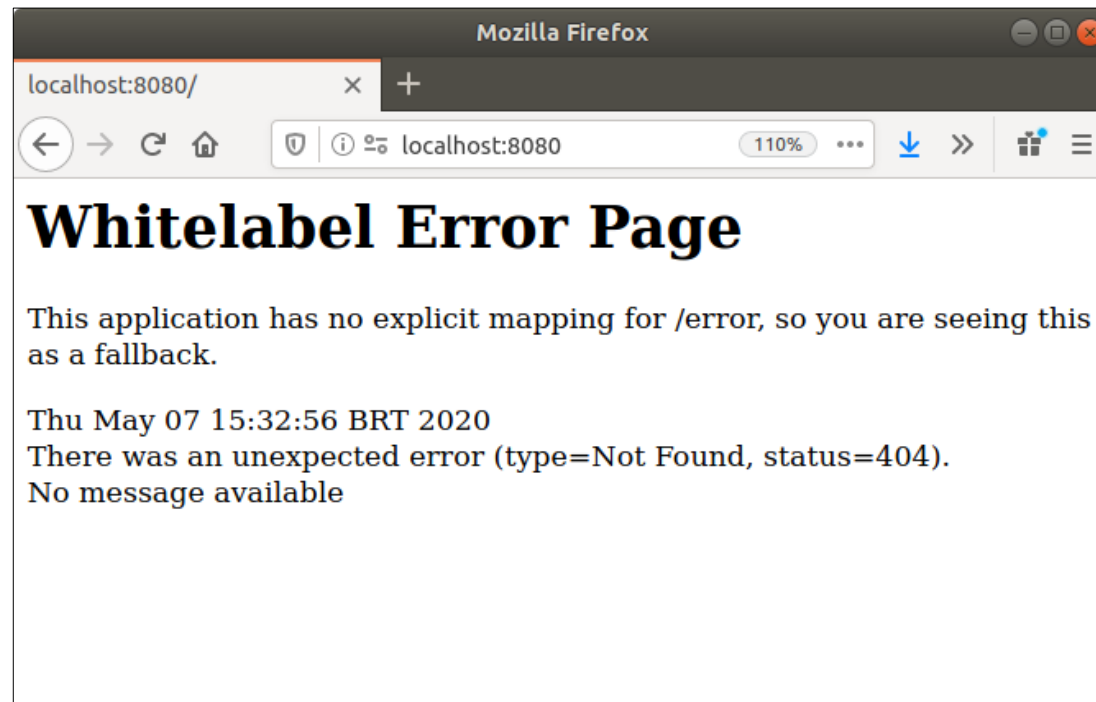
:: Spring Boot ::                (v2.2.6.RELEASE)

2020-05-07 15:31:39.795 INFO 18448 --- [main] com.example.demo.DemoApplication : No active p
2020-05-07 15:31:39.798 INFO 18448 --- [main] com.example.demo.DemoApplication : Tomcat init
2020-05-07 15:31:40.505 INFO 18448 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Starting se
2020-05-07 15:31:40.512 INFO 18448 --- [main] o.apache.catalina.core.StandardService : Starting Se
2020-05-07 15:31:40.513 INFO 18448 --- [main] org.apache.catalina.core.StandardEngine : Starting Se
2020-05-07 15:31:40.556 INFO 18448 --- [main] o.a.c.c.C.[Tomcat].[/] : Initializin
2020-05-07 15:31:40.556 INFO 18448 --- [main] o.s.web.context.ContextLoader : Root WebApp
2020-05-07 15:31:40.692 INFO 18448 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializin
2020-05-07 15:31:40.806 INFO 18448 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat star
2020-05-07 15:31:40.809 INFO 18448 --- [main] com.example.demo.DemoApplication : Started Dem
2020-05-07 15:32:56.381 INFO 18448 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[/] : Initializin
2020-05-07 15:32:56.381 INFO 18448 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializin
2020-05-07 15:32:56.385 INFO 18448 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed i
```

Tomcat embutido  
no Spring

# Executando a aplicação no Eclipse (6/6)

- Se você digitar localhost:8080 em um browser, vai obter a página padrão de erro do Spring

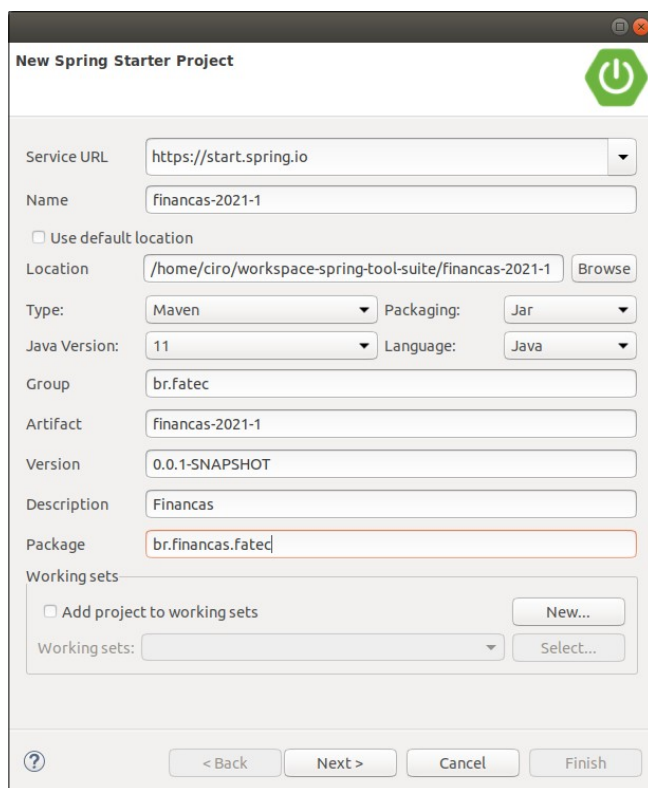


# Spring Tool Suite (1/3)

- <https://spring.io/tools>  Spring Tool Suite | 4
- Eclipse configurado para Spring
- Também é possível instalar um plugin do Spring no Eclipse

# Spring Tool Suite (2/3)

- Para criar um projeto no STS
  - **File** → **New** → **Spring Starter Project**



**New Spring Starter Project**

Service URL:

Name:

☐ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

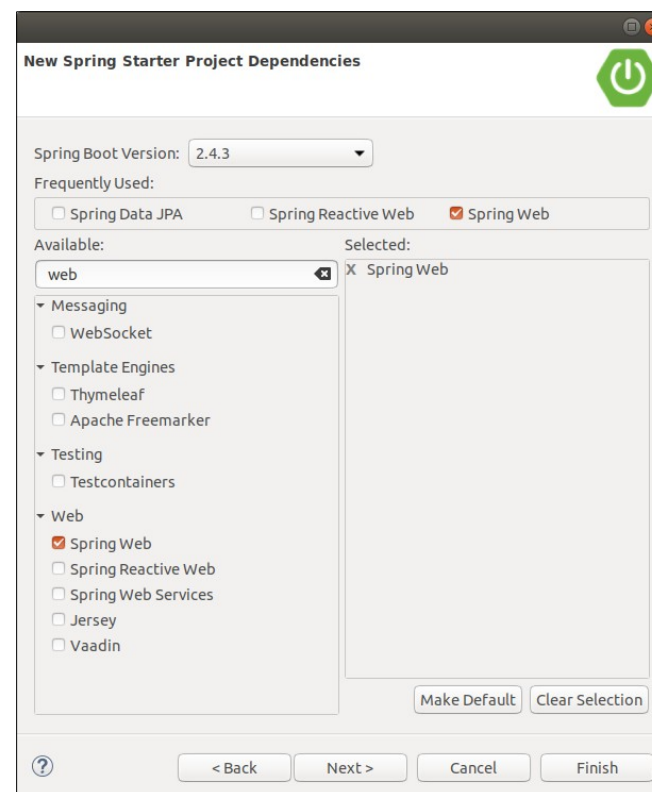
Description:

Package:

Working sets:

☐ Add project to working sets

Working sets:



**New Spring Starter Project Dependencies**

Spring Boot Version:

Frequently Used:

☐ Spring Data JPA ☐ Spring Reactive Web ☒ Spring Web

Available:

☒ web

Selected:

☒ Spring Web

# Spring Tool Suite (3/3)

- Para executar o projeto
  - Clique com o botão direito do mouse sobre o projeto
  - Selecione a opção **Run as** → **Spring Boot App**

# Criando um Web Service REST (1/5)

- Para criar uma classe de recursos REST basta anotá-la com `@RestController`
- Para definir o caminho (path) do recurso usamos a anotação `@RequestMapping("/path")`
- Os métodos de recursos podem ser anotadas com
  - `@GetMapping`
  - `@PostMapping`
  - `@PutMapping`
  - `@DeleteMapping`



# Criando um Web Service REST (2/5)

## ■ Parâmetros

- Elemento `value`: usado para definir a rota para uma método específico
  - `@GetMapping(value = "/foos")`
  - Também pode ser usado para definir valores que são passados na URL da rota
  - `@GetMapping("/foos/{id}")`
- `@PathVariable`: usado para associar um valor da rota da requisição a um argumento do método
  - ```
public ResponseEntity<?>  
get(@PathVariable("id") Long id)
```

**`http://localhost:8080/spring-mvc-basics/api/foos/123`**

# Criando um Web Service REST (3/5)

- ... Parâmetros

- `@RequestParam`: usado para associar um parâmetro da requisição a um argumento do método

```
GetMapping("/foos")
```

```
public ResponseEntity<?> get(@RequestParam("id")  
                             Long id) { ... }
```

**`http://localhost:8080/spring-mvc-basics/api/foos?id=123`**

# Criando um Web Service REST (4/5)

- Valores no corpo da requisição
  - `@RequestBody`: usado para associar um objeto passado no corpo da requisição a um parâmetro de um método
    - ```
public ResponseEntity<?> post (@RequestBody  
                                Conta conta) {
```

# Criando um Web Service REST (5/5)

---

- `ResponseEntity`
  - Representa a resposta HTTP: status code, headers e body
  - Pode ser usada para configurar completamente a resposta HTTP



# Conta.java

```
package br.financas.fatec.model;
import java.io.Serializable;

public class Conta implements Serializable {
    private static final long serialVersionUID = 1L;

    private static Long nextId = 1L;
    private Long id;
    private Integer agencia;
    private String numero;
    private String titular;
    private Float saldo;

    public Conta() { }

    public Conta(Long id) { this.id = id; }

    public Long generateId() {
        return nextId++;
    }
    // getters e setters
    // hashCode e equals
}
```

# ContaService.java (1/3)

- Classe que simula a persistência dos dados

```
package br.financas.fatec.service;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import org.springframework.stereotype.Service;
```

```
import br.financas.fatec.model.Conta;
```

```
@Service
```

```
public class ContaService {
```

```
    private static List<Conta> contas = new ArrayList<>();
```

```
    public ContaService() {}
```

```
    public void create(Conta conta) {  
        conta.setId(conta.generateId());  
        contas.add(conta);  
    }
```

# ContaService.java (2/3)

```
public List<Conta> findAll() {  
    return contas;  
}  
  
public Conta find(Conta conta) {  
    for (Conta c : contas) {  
        if (c.equals(conta)) {  
            return c;  
        }  
    }  
    return null;  
}  
  
public Conta find(Long id) {  
    return find(new Conta(id));  
}
```

# ContaService.java (3/3)

```
public boolean update(Conta conta) {
    Conta _conta = find(conta);
    if (_conta != null) {
        _conta.setAgencia(conta.getAgencia());
        _conta.setNumero(conta.getNumero());
        _conta.setTitular(conta.getTitular());
        _conta.setSaldo(conta.getSaldo());
        return true;
    }
    return false;
}

public boolean delete(Long id) {
    Conta _conta = find(id);
    if (_conta != null) {
        contas.remove(_conta);
        return true;
    }
    return false;
}
}
```



# ContaController.java (1/3)

```
package br.financas.fatec.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import br.financas.fatec.model.Conta;
import br.financas.fatec.service.ContaService;
```

```
@RestController
```

```
@RequestMapping("/contas")
```

```
public class ContaController {
```

# ContaController.java (2/3)

```
@Autowired  
private ContaService service;
```

```
@GetMapping  
public ResponseEntity<List<Conta>> getAll() {  
    return ResponseEntity.ok(service.findAll());  
}
```

```
@GetMapping(value =("/{id}")  
public ResponseEntity<?> get(@PathVariable("id") Long id) {  
    Conta _conta = service.find(id);  
    if (_conta != null)  
        return ResponseEntity.ok(_conta);  
    return ResponseEntity.status(HttpStatus.NOT_FOUND).build();  
}
```

```
@PostMapping  
public ResponseEntity<Conta> post(@RequestBody Conta conta) {  
    service.create(conta);  
    return ResponseEntity.ok(conta);  
}
```

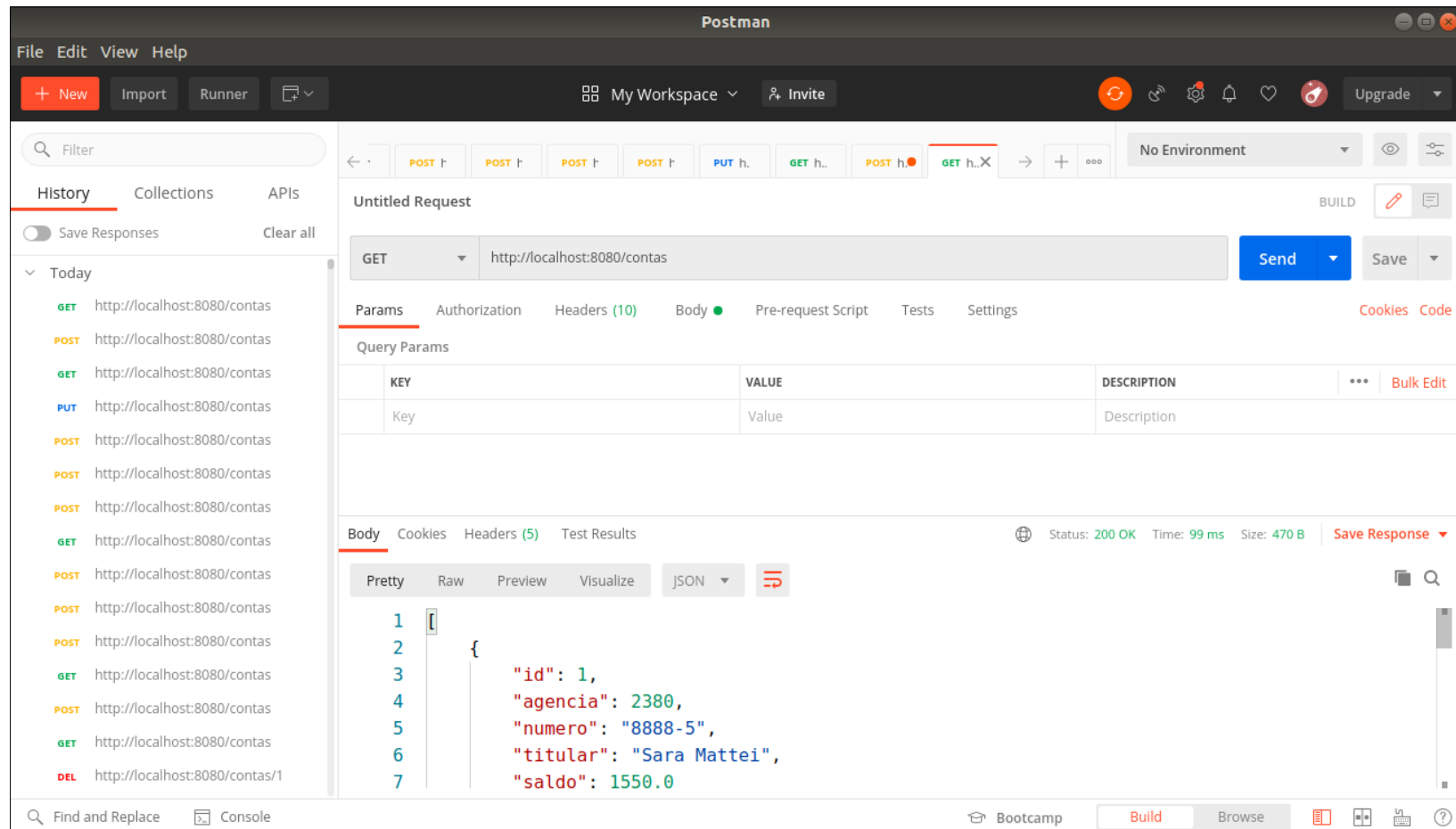
# ContaController.java (3/3)

```
@PostMapping
public ResponseEntity<?> put(@RequestBody Conta conta) {
    if (service.update(conta)) {
        return ResponseEntity.ok(conta);
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
}

@DeleteMapping(value =("/{id}")
public ResponseEntity<?> delete(@PathVariable("id") Long id) {
    if (service.delete(id)) {
        return ResponseEntity.ok().build();
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
}

}
```

# Testando a API com o Postman



The screenshot shows the Postman application interface. The top bar includes the menu (File, Edit, View, Help), a toolbar with buttons for New, Import, Runner, and a workspace selector set to 'My Workspace'. The left sidebar shows a 'History' tab with a list of recent requests, including GET and POST requests to 'http://localhost:8080/contas'. The main panel displays an 'Untitled Request' for a GET method to 'http://localhost:8080/contas'. The 'Params' tab is active, showing a table for Query Params with columns KEY, VALUE, and DESCRIPTION. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format:

```

1 [
2   {
3     "id": 1,
4     "agencia": 2380,
5     "numero": "8888-5",
6     "titular": "Sara Mattei",
7     "saldo": 1550.0
  }
]

```

The status bar at the bottom indicates a successful response with status 200 OK, time 99 ms, and size 470 B. The bottom of the interface includes a 'Find and Replace' search bar, a 'Console' tab, and a 'Bootcamp' section with 'Build' and 'Browse' buttons.

# Referência

- Spring Boot. Disponível em:  
<https://spring.io/projects/spring-boot>