

SPRING E CRUD REST

ATIVIDADES DE HOJE

Horário das atividades

- 13h30 Faísca
- 13h45 Spring e CRUD REST
- 15h00 Intervalo
- 15h15 Exercícios
- 16h40 Checkout
- 17h00 Chamada



SPRING

Uma breve recapitulação

O que é o Spring?

- Spring é descrito como um framework de estrutura leve para a construção de aplicativos Java. Porém, essa definição traz dois pontos interessantes.
- **Primeiro**, é que o Spring pode ser utilizado para construir qualquer aplicativo em Java (por exemplo, aplicativos autônomos, web ou JEE). Ao contrário de outras estruturas, como por exemplo, o Apache Struts, que é um framework limitado a aplicativos web.
- **Segundo**, a leveza que define a filosofia do Spring, no sentido de que é necessário fazer poucas, se houver, mudanças no código da aplicação para obter os benefícios do Spring Core.

Características do Spring

- Com o Spring, não precisamos de um servidor de aplicação para funcionar. Pois, pode rodar usando apenas a JVM;
- O Spring permite utilizar apenas aquilo que é necessário para o projeto. Como citado anteriormente, o J2EE nos levava a implementar comportamentos que não eram necessários, fazendo com que a aplicação levasse muito mais do que era necessário;
- O core do Spring é baseado no princípio de inversão de controle e injeção de dependência, se responsabilizando em criar e gerenciar os componentes da aplicação, os quais chamamos de beans.

Inversão de Controle (IoC) e Injeção de dependência (DI)

- O pattern de **Inversão de Controle** permite delegar a outro elemento o controle sobre como e quando um objeto deve ser criado e quando um método deve ser executado. Assim, essas responsabilidades de controle sobre a execução de alguns comportamentos, passa a ser gerenciado por esse elemento, não cabendo mais a nós, programadores.
- Na **Injeção de Dependência** a classe deixa de se preocupar em como resolver as suas dependências. Ela apenas mantém o foco no uso dos recursos para realizar as tarefas que precisa. Consequentemente, isso nos leva a uma das características mais conhecidas do universo Spring. Podemos delegar o gerenciamento das classes para o Framework e com isso, não precisamos utilizar o **new** para criar os objetos.

Ecosistema do Spring



Ecossistema do Spring

Alguns dos projetos (módulos) do Framework Spring

- **Spring Boot** - módulo produtivo do Spring que impulsionou o desenvolvimento de microsserviços, além de ajudar na configuração importando e configurando automaticamente todas as dependências.
- **Spring Security** - módulo usado para adicionarmos autenticação na aplicação.
- **Spring Batch** - módulo usado caso haja necessidade de criar muitos processos com horários agendado, permitindo o desenvolvimento de aplicativos de lote robustos.
- **Spring Data** - a missão do módulo é fornecer um modelo de programação familiar e consistente para acesso a dados, enquanto ainda retém as características especiais do armazenamento de dados subjacente.
- **Spring Cloud** - módulo que fornece ferramentas para que os desenvolvedores criem rapidamente alguns dos padrões comuns em sistemas distribuídos

Spring Initializr

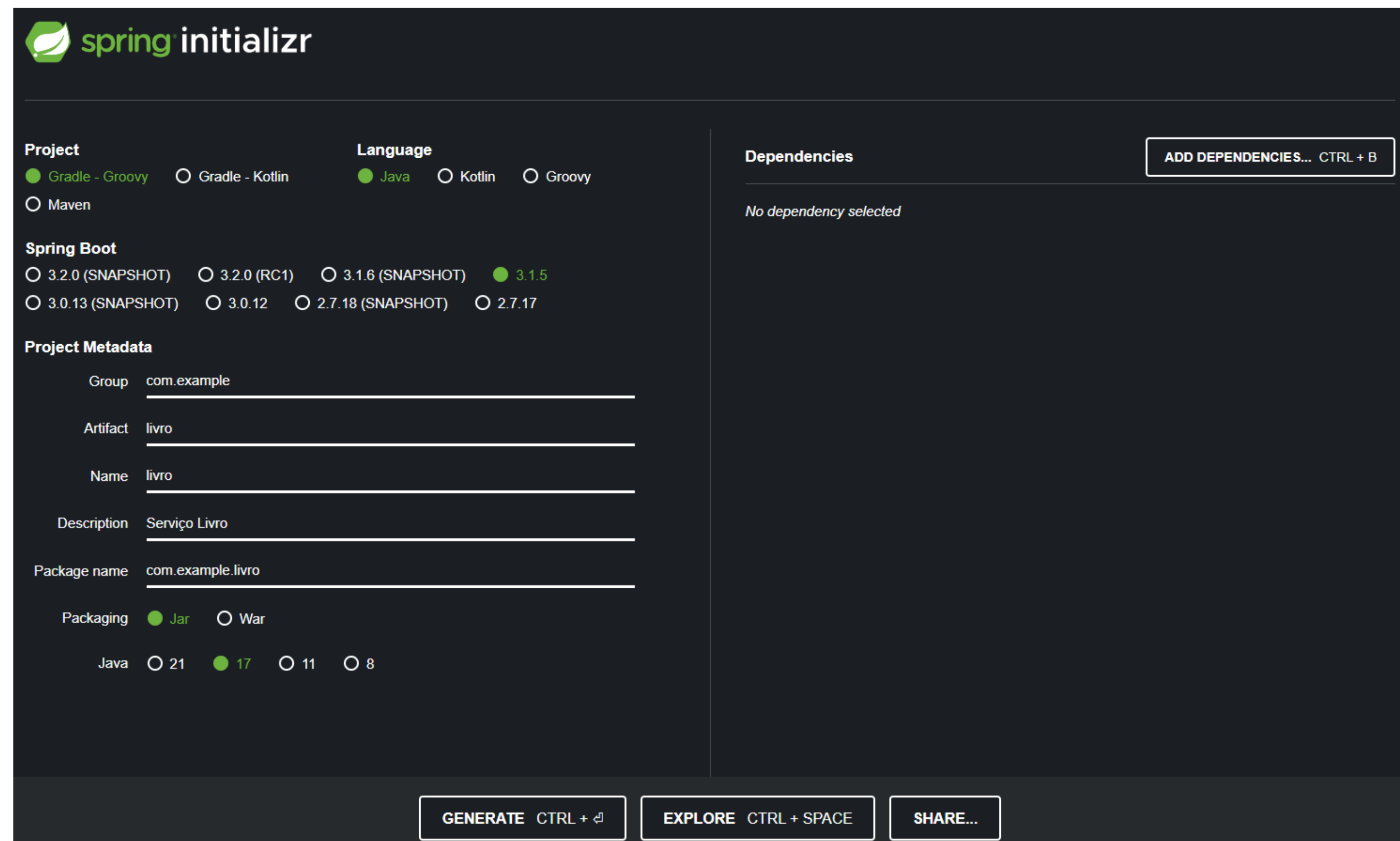
O que é Spring Initializr?

- O Spring Initializr é uma ferramenta que fornece uma interface web bem simples para o usuário.
- Podendo gerar seu projeto a partir de uma estrutura de configurações pré-moldadas.
- São configurações de versões do java/spring boot, grupo/nome do projeto, série de lista de dependências e etc.

Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

1. Acessar a interface web do Spring Initializr

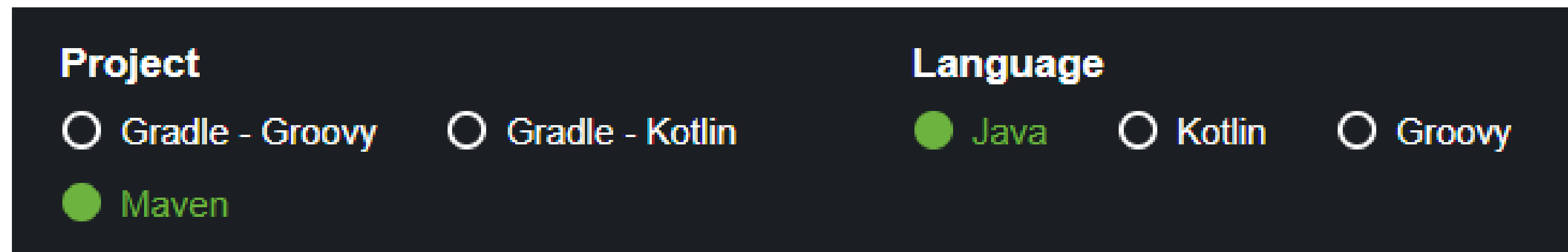


The screenshot shows the Spring Initializr web interface. The header features the 'spring initializr' logo. The main content area is divided into three sections: 'Project', 'Language', and 'Dependencies'. The 'Project' section includes radio buttons for 'Gradle - Groovy' (selected), 'Gradle - Kotlin', and 'Maven'. The 'Language' section includes radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section includes radio buttons for various versions, with '3.1.5' selected. The 'Project Metadata' section includes input fields for 'Group' (com.example), 'Artifact' (livro), 'Name' (livro), 'Description' (Serviço Livro), and 'Package name' (com.example.livro). The 'Packaging' section includes radio buttons for 'Jar' (selected) and 'War'. The 'Java' section includes radio buttons for versions 21, 17 (selected), 11, and 8. The 'Dependencies' section includes a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

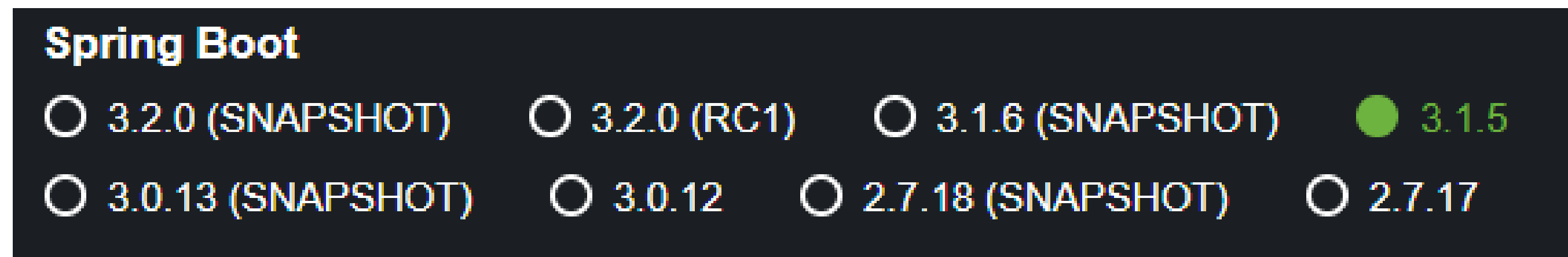
2. Criaremos um projeto na linguagem Java, utilizando o *Maven*.



The screenshot shows the 'Project' and 'Language' selection options in the Spring Initializr. Under 'Project', 'Maven' is selected with a green dot, while 'Gradle - Groovy' and 'Gradle - Kotlin' are unselected. Under 'Language', 'Java' is selected with a green dot, while 'Kotlin' and 'Groovy' are unselected.

Project		Language		
<input type="radio"/> Gradle - Groovy	<input type="radio"/> Gradle - Kotlin	<input checked="" type="radio"/> Java	<input type="radio"/> Kotlin	<input type="radio"/> Groovy
<input checked="" type="radio"/> Maven				

3. Usaremos a versão 3.1.5 do Spring



The screenshot shows the 'Spring Boot' version selection options in the Spring Initializr. The version '3.1.5' is selected with a green dot. Other versions shown include '3.2.0 (SNAPSHOT)', '3.2.0 (RC1)', '3.1.6 (SNAPSHOT)', '3.0.13 (SNAPSHOT)', '3.0.12', '2.7.18 (SNAPSHOT)', and '2.7.17'.

Spring Boot			
<input type="radio"/> 3.2.0 (SNAPSHOT)	<input type="radio"/> 3.2.0 (RC1)	<input type="radio"/> 3.1.6 (SNAPSHOT)	<input checked="" type="radio"/> 3.1.5
<input type="radio"/> 3.0.13 (SNAPSHOT)	<input type="radio"/> 3.0.12	<input type="radio"/> 2.7.18 (SNAPSHOT)	<input type="radio"/> 2.7.17

Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

Agora, vamos preencher alguns dados do metadata do projeto.

Project Metadata

Group	<input type="text" value="com.example"/>
Artifact	<input type="text" value="livro"/>
Name	<input type="text" value="livro"/>
Description	<input type="text" value="Serviço Livro"/>
Package name	<input type="text" value="com.example.livro"/>
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 21 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

Como gerar uma aplicação?

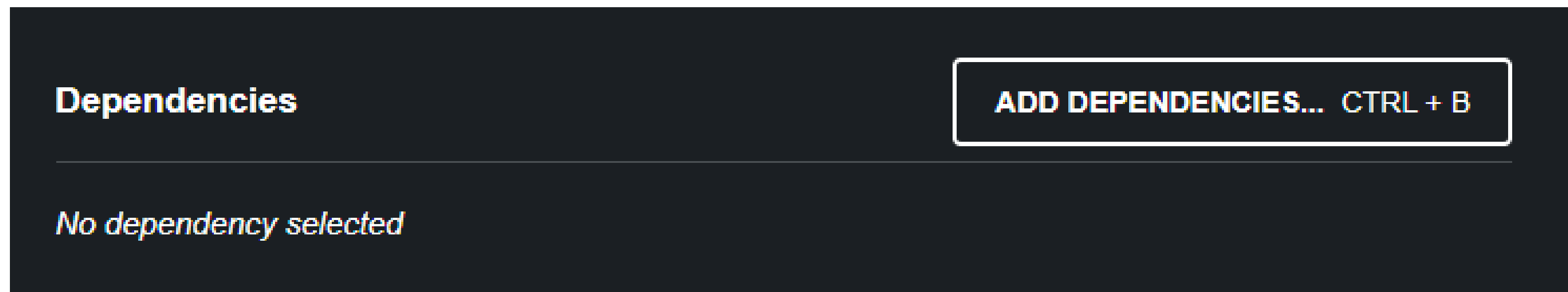
Para gerar uma aplicação, você deve seguir os seguintes passos:

- **Group** - geralmente o domínio reverso da empresa ou organização. Aqui colocaremos br.com.artigo (você pode colocar o que achar melhor)
- **Artifact** - o artefato a ser gerado (nome da aplicação). Aqui colocaremos livro
- **Name** - será automaticamente preenchido com o mesmo valor do campo Artifact, no caso livro. Pode ser mudado, mas manteremos o padrão.
- **Description** - uma rápida descrição do seu projeto.
- **Package name** - estrutura do pacote inicial da aplicação.
- **Packaging** - como se.rá empacotada a aplicação, deixaremos o padrão **Jar**.
- **Java** - versão do java utilizada. Aqui eu selecionei o **Java 17**.

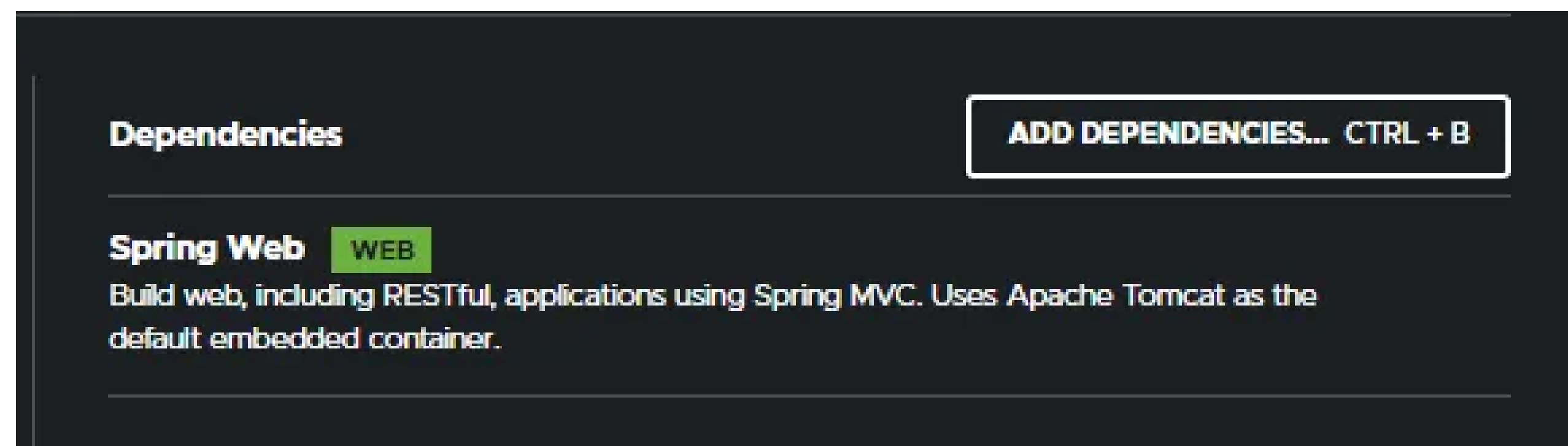
Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

4. Adicionando dependências no projeto.



Clicando em ADD DEPENDENCIES, digite Spring Web.



A dependência **Spring Web** nos permite criar aplicativos da web, incluindo RESTful, usando Spring MVC. Usa Apache Tomcat como o contêiner embutido padrão.

Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

Adicionamos a dependência (opcional) Lombok para nos ajudar com Construtores, Getters, Setters, etc.



Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

Adicionamos a dependência (opcional) Lombok para nos ajudar com Construtores, Getters, Setters, etc.



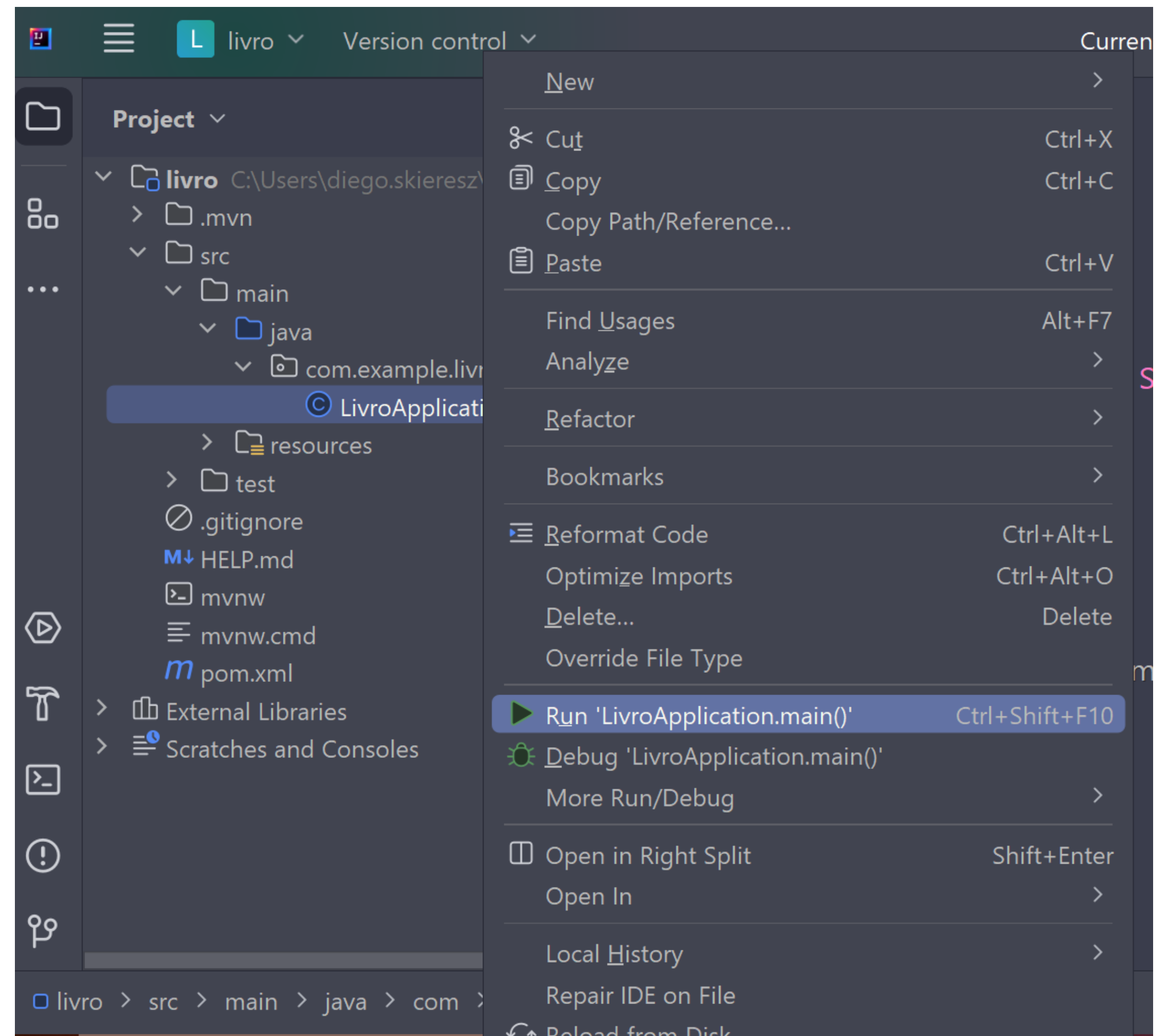
5. Basta agora, gerar o projeto clicando no botão *Generate*. O artefato do projeto será baixado em formato zip.

Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

6. Vá para File -> Open e abra o projeto baixado. O IntelliJ deve reconhecer automaticamente a ferramenta de construção e criar, importar as dependências necessárias e configurá-la para você.

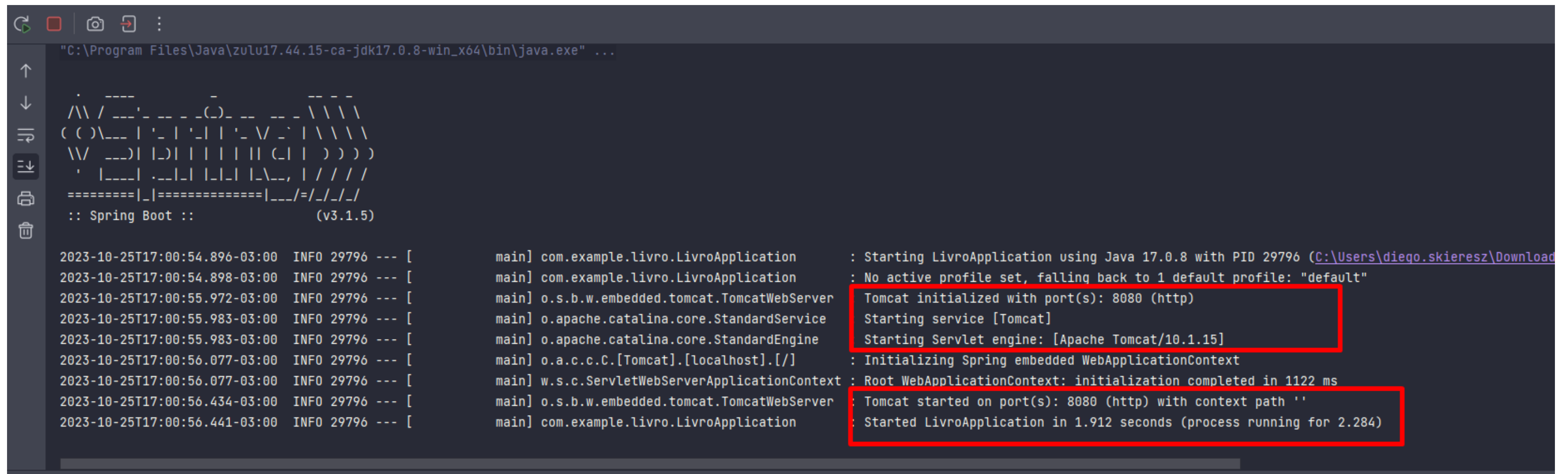
7. *Classe Main — encontre a classe main, **LivroApplication**. O Spring Initializr já preparou ela para nós! Basta dar um run e o projeto estará rodando!*



Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

O Spring Boot utiliza o Tomcat por por debaixo dos panos. E inicializa, por padrão, nossa aplicação na porta 8080.



```
"C:\Program Files\Java\zu1u17.44.15-ca-jdk17.0.8-win_x64\bin\java.exe" ...

  ____ _
 / ___ \| | | |
/ /   \| |_| |
\ \   /| | | |
 \___/\|_|_|_|

:: Spring Boot ::                (v3.1.5)

2023-10-25T17:00:54.896-03:00 INFO 29796 --- [main] com.example.livro.LivroApplication : Starting LivroApplication using Java 17.0.8 with PID 29796 (C:\Users\diego.skieresz\Download
2023-10-25T17:00:54.898-03:00 INFO 29796 --- [main] com.example.livro.LivroApplication : No active profile set, falling back to 1 default profile: "default"
2023-10-25T17:00:55.972-03:00 INFO 29796 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-10-25T17:00:55.983-03:00 INFO 29796 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-10-25T17:00:55.983-03:00 INFO 29796 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.15]
2023-10-25T17:00:56.077-03:00 INFO 29796 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-10-25T17:00:56.077-03:00 INFO 29796 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1122 ms
2023-10-25T17:00:56.434-03:00 INFO 29796 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-10-25T17:00:56.441-03:00 INFO 29796 --- [main] com.example.livro.LivroApplication : Started LivroApplication in 1.912 seconds (process running for 2.284)
```


Como gerar uma aplicação?

Para gerar uma aplicação, você deve seguir os seguintes passos:

8. Acessando nossa aplicação em <http://localhost:8080/>

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Oct 25 17:04:27 BRT 2023

There was an unexpected error (type=Not Found, status=404).

Não se preocupe se você chegou a esta página, o que significa que seu aplicativo está funcionando perfeitamente. Nada aparece pois ainda não programamos nada! Mas o servidor está lá! Rodando!

O que é o Controller?

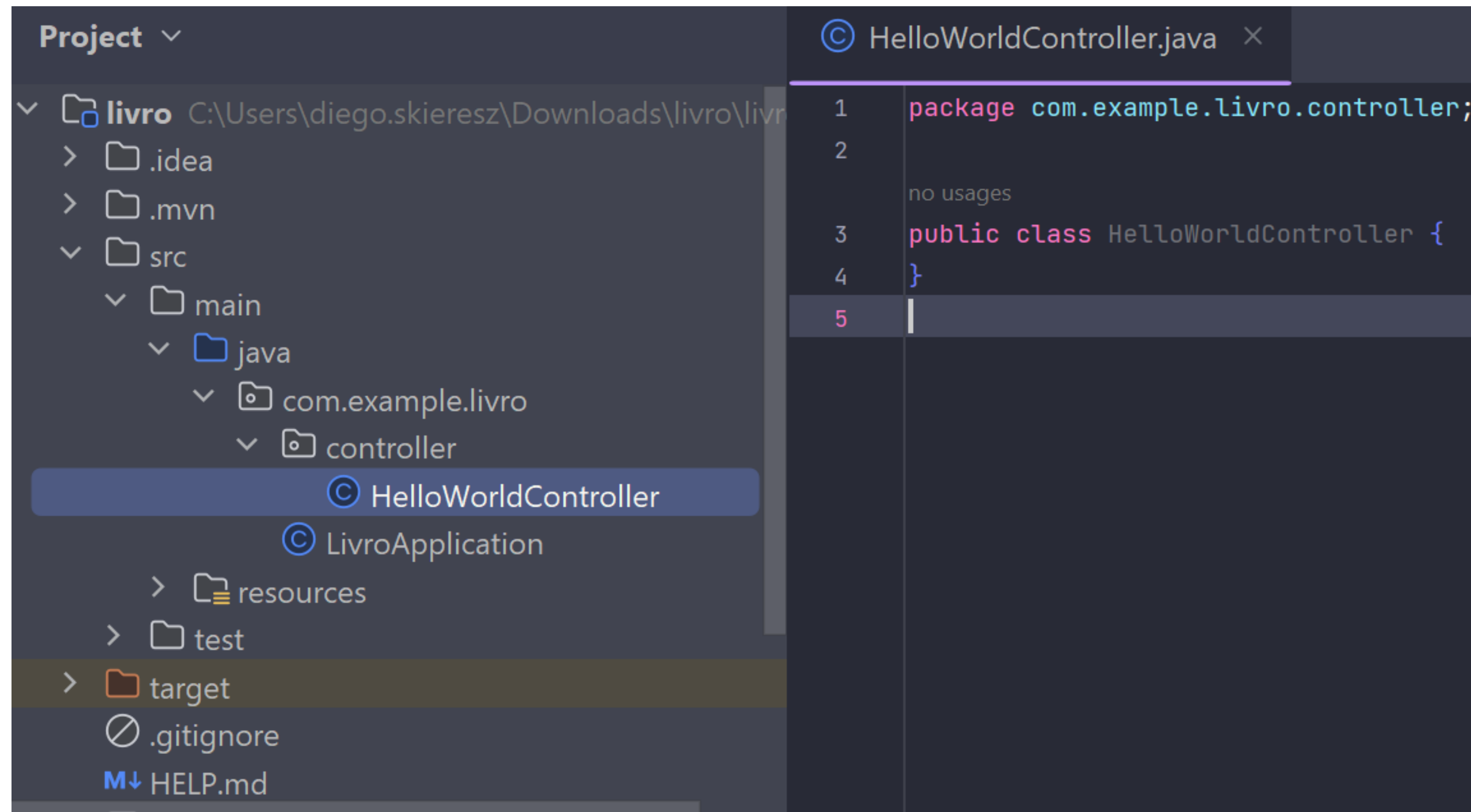
- O Controller é a linha de frente do nosso projeto, a porta de entrada.
- Ele é o responsável por controlar as requisições recebidas indicando quem deve atender as requisições e para quem deve responde-las.



Nosso primeiro Controller



- Crie uma classe com o nome de **HelloWorldController**



The screenshot shows an IDE with a project named 'livro' located at 'C:\Users\diego.skieresz\Downloads\livro\livro'. The project structure is as follows:

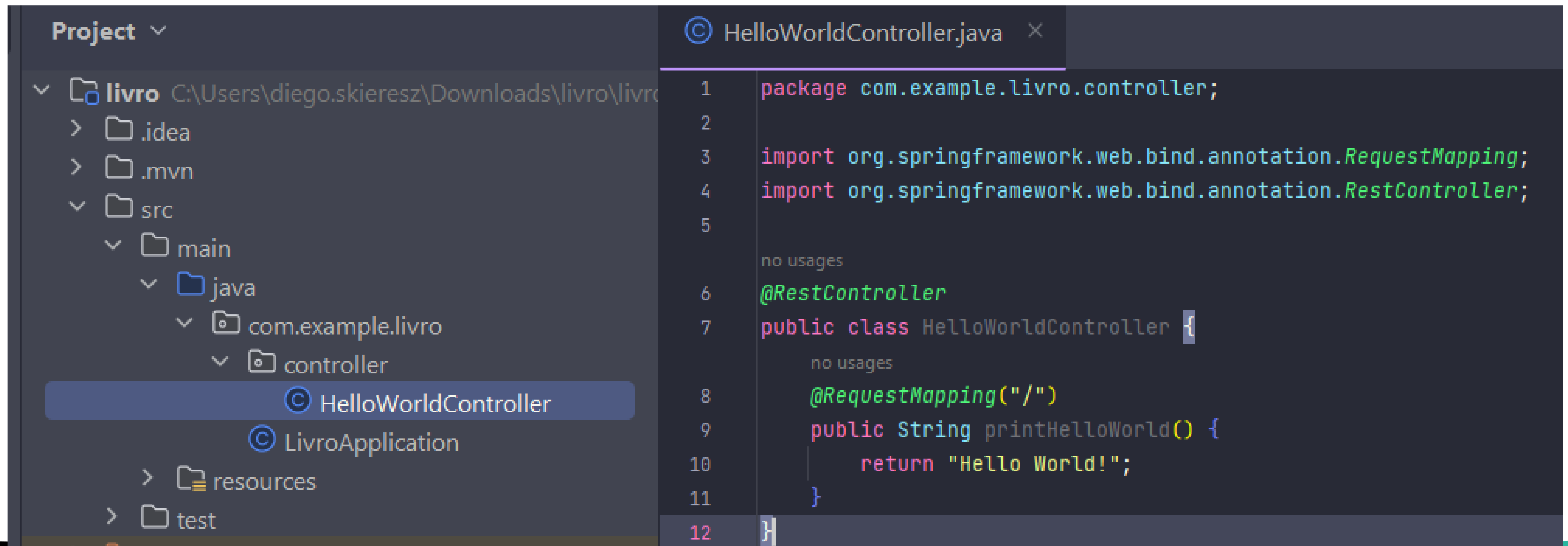
- livro
 - .idea
 - .mvn
 - src
 - main
 - java
 - com.example.livro
 - controller
 - HelloWorldController (selected)
 - LivroApplication
 - resources
 - test
 - target
 - .gitignore
 - HELP.md

The 'HelloWorldController.java' file is open in the editor, showing the following code:

```
1 package com.example.livro.controller;  
2  
3 no usages  
4 public class HelloWorldController {  
5 }  
6
```

Nosso primeiro Controller

- Crie um mapeamento de caminho **REST** para “/”. Que quando visitado exibirá a String retornada (Hello World). Você pode retornar qualquer string que desejar do método



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure is as follows:

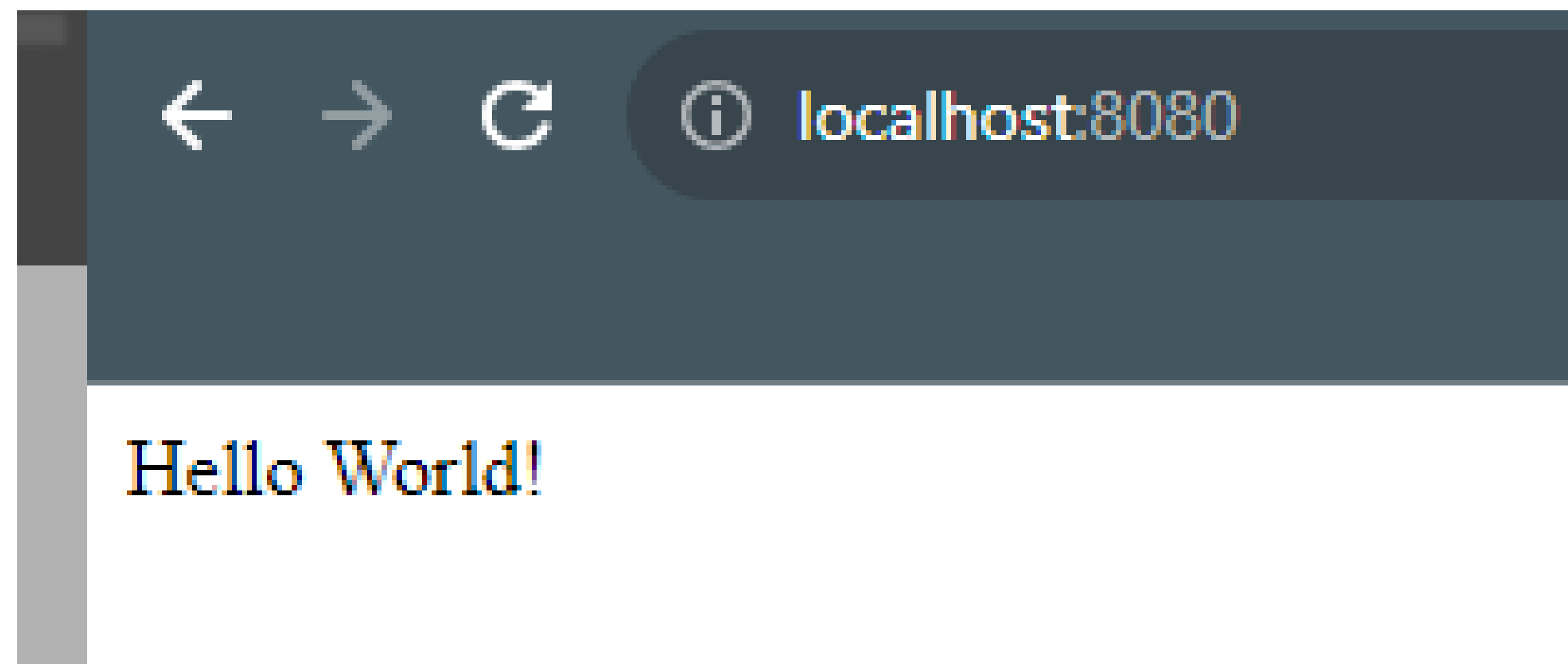
- Project
 - livro C:\Users\diego.skieresz\Downloads\livro\livro
 - .idea
 - .mvn
 - src
 - main
 - java
 - com.example.livro
 - controller
 - © HelloWorldController (selected)
 - © LivroApplication
 - resources
 - test

The code editor shows the following Java code for `HelloWorldController.java`:

```
1 package com.example.livro.controller;  
2  
3 import org.springframework.web.bind.annotation.RequestMapping;  
4 import org.springframework.web.bind.annotation.RestController;  
5  
6 no usages  
7 @RestController  
8 public class HelloWorldController {  
9     no usages  
10     @RequestMapping("/")  
11     public String printHelloWorld() {  
12         return "Hello World!";  
13     }  
14 }
```

Nosso primeiro Controller

- Rode a aplicação e chame a URL que o nosso controller está “escutando”
localhost:8080



Refatorando

Lembra que o Controller é a porta de entrada para nossa aplicação?

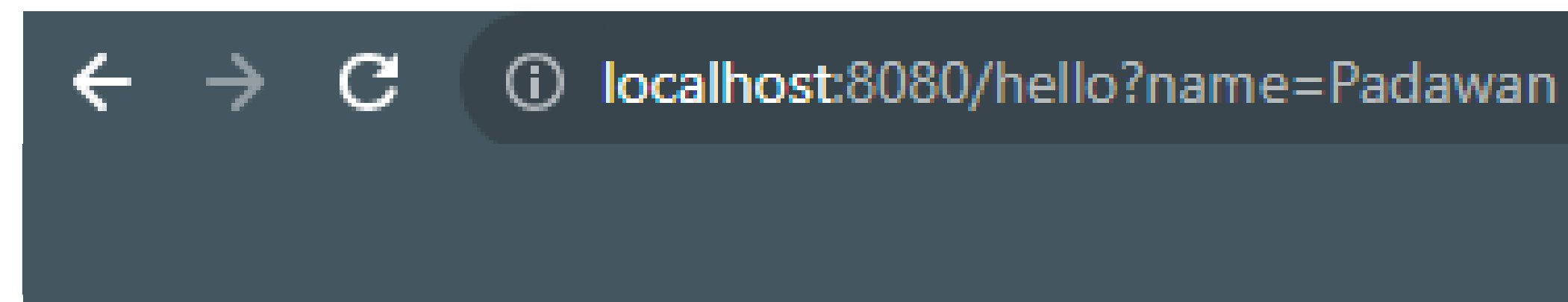
- A anotação `@GetMapping` define que o nosso método `hello` é responsável por tratar tudo que chega na URL `/hello` com o verbo `http GET`.
- O Controller procura na URL por uma variável com o nome `"name"`. Com o valor padrão de `"World"`
- Passamos a variável `"name"` na URL e vamos ver como nosso Controller se comporta.

```
© HelloWorldController.java x
1  package com.example.livro.controller;
2
3  import org.springframework.web.bind.annotation.GetMapping;
4  import org.springframework.web.bind.annotation.RequestParam;
5  import org.springframework.web.bind.annotation.RestController;
6
7  no usages
8  @RestController
9  public class HelloWorldController {
10     no usages
11     @GetMapping("/hello")
12     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
13         return String.format("Hello %s!", name);
14     }
15 }
```

Nosso primeiro Controller

Modificando o código

- *localhost:8080/hello?name=Padawan*



Hello Padawan!

Quer saber mais sobre anotações?

- **@RequestMapping** - Ele fornece o mapeamento entre o caminho da solicitação e o método manipulador.
- **@RequestParam** - Usada para mapear os parâmetros HTTP a argumentos de métodos. Por exemplo, se você enviar parâmetros de consulta juntamente com o URL para paginação.
- **@SpringBootApplication** - Essa annotation única combina três anotações como **@Configuration**, **@EnableAutoConfiguration** e **@ComponentScan**.
- <https://www.java67.com/2019/04/top-10-spring-mvc-and-rest-annotations-examples-java.html>

Postman

Porque Postman?

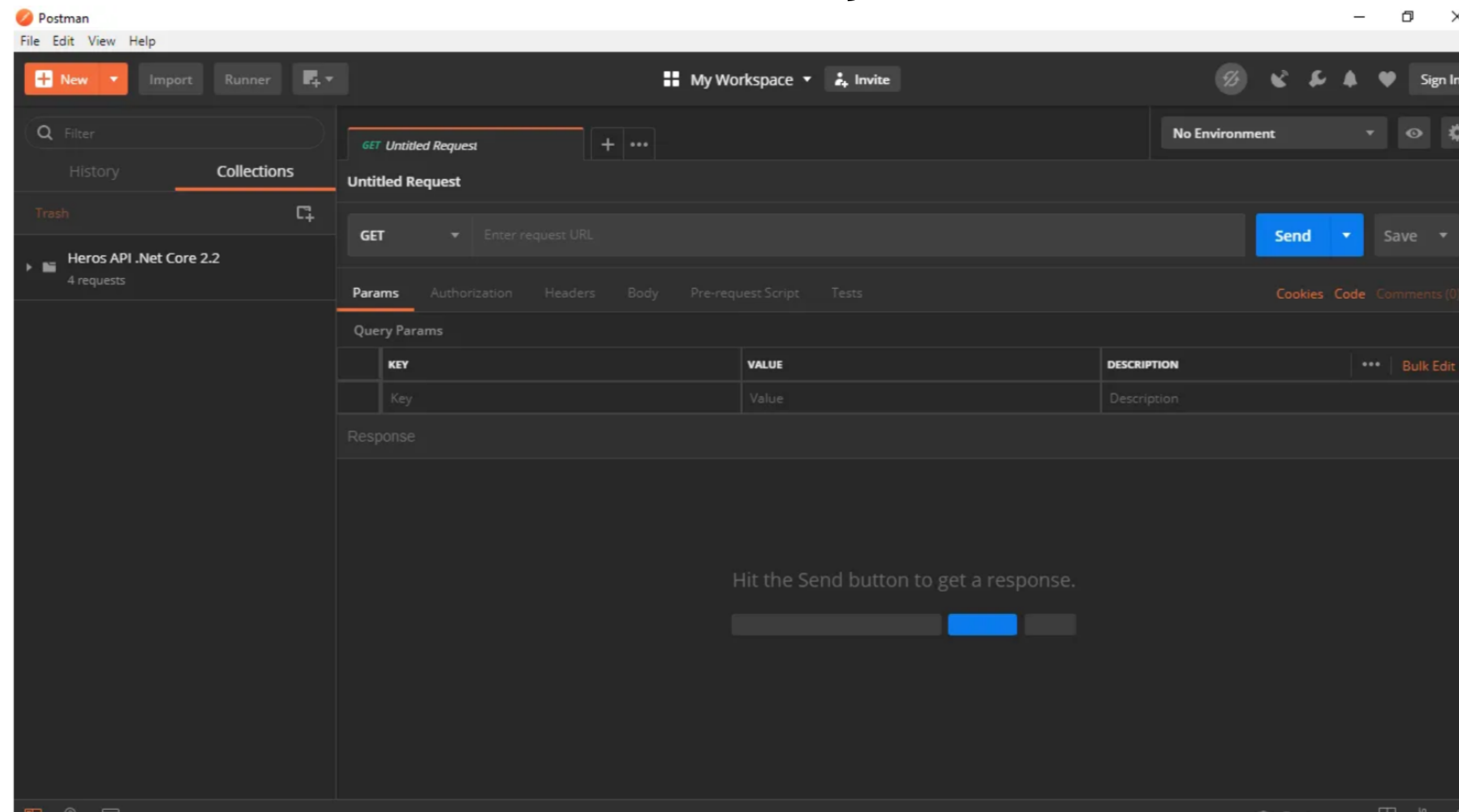


- **Postman** é uma ótima ferramenta para testar sua API enviando solicitações HTTP e visualizando as respostas.
- **Testes automatizados**, testes de API **GraphQL** e integração com ferramentas populares de **CI/CD**, como **Jenkins**, **GitLab** e **Travis CI**.
- Você pode usar o **Postman** para criar **solicitações**, organizá-las em **coleções** e **adicionar testes** para verificar as **respostas** da sua API.
- Você também pode criar **ambientes** para armazenar diferentes **configurações** de endpoints de API e alternar entre eles facilmente.

Coleções

Grupos de solicitações de API organizadas e armazenadas juntas.

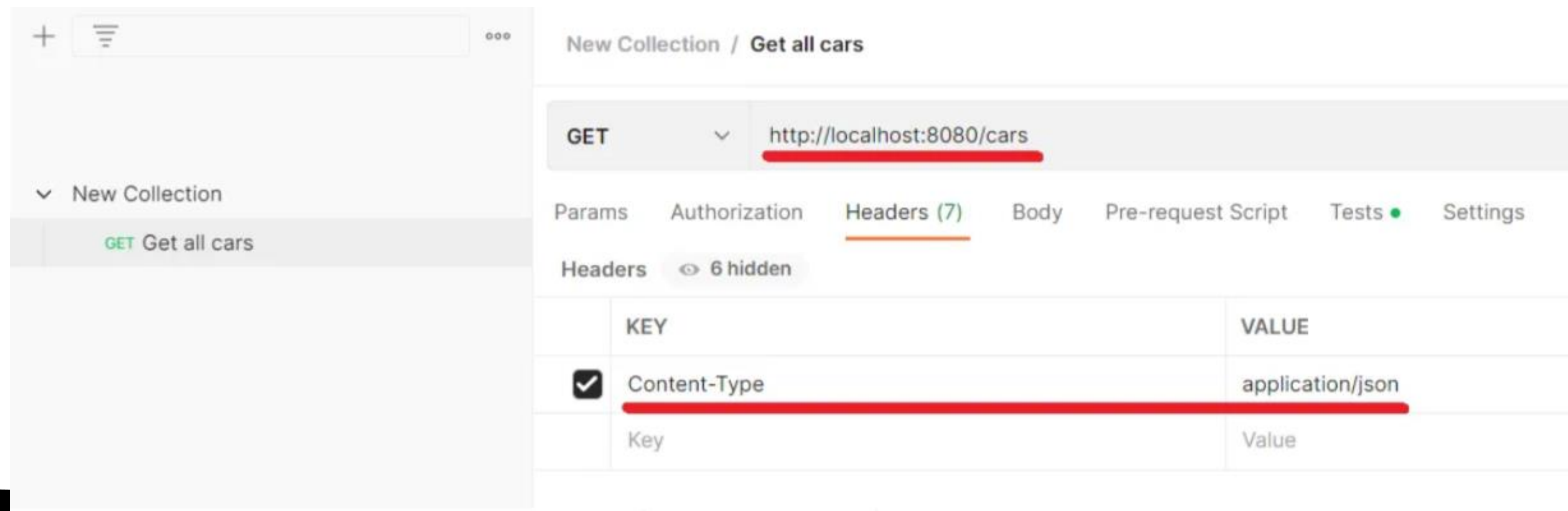
- Clique no canto superior esquerdo em New, em seguida selecione a opção Collection e defina um nome e uma descrição.



Adicionando Request

Representação de uma URL do endpoint da API, método HTTP que pode ser enviado para a API

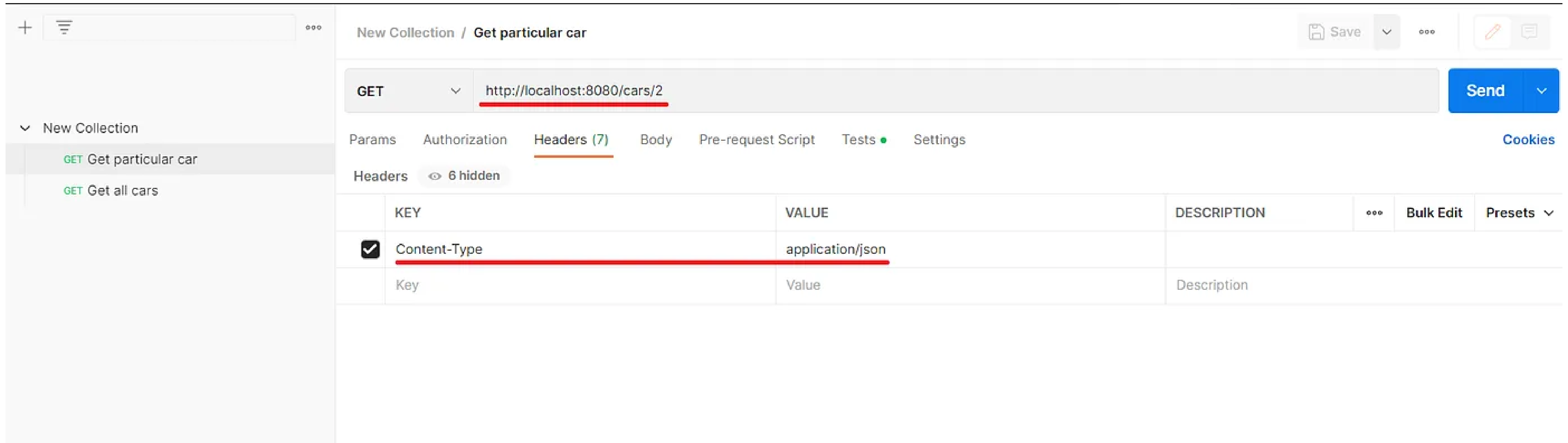
- Clique no botão "Adicionar requisição" no lado direito do nome da sua Coleção.
- Dê um nome para sua requisição, selecione o método HTTP (por exemplo, GET, POST, PUT, DELETE).
- Insira o endpoint da API e quaisquer cabeçalhos de requisição necessários.
- Salve a requisição clicando no botão "Salvar" no canto superior direito.



Adicionando Request

Representação de uma URL do endpoint da API, método HTTP que pode ser enviado para a API

- Outro exemplo de GET agora trazendo um ID específico



The screenshot shows a REST client interface with a sidebar on the left and a main panel on the right. The sidebar contains a 'New Collection' dropdown and two items: 'GET Get particular car' and 'GET Get all cars'. The main panel is titled 'New Collection / Get particular car' and features a 'Send' button. The request method is 'GET' and the URL is 'http://localhost:8080/cars/2'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Headers' tab is active, showing a table with headers. The first header is 'Content-Type' with a value of 'application/json'. Below this, there is a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Content-Type	application/json	
Key	Value	Description

Adicionando Request

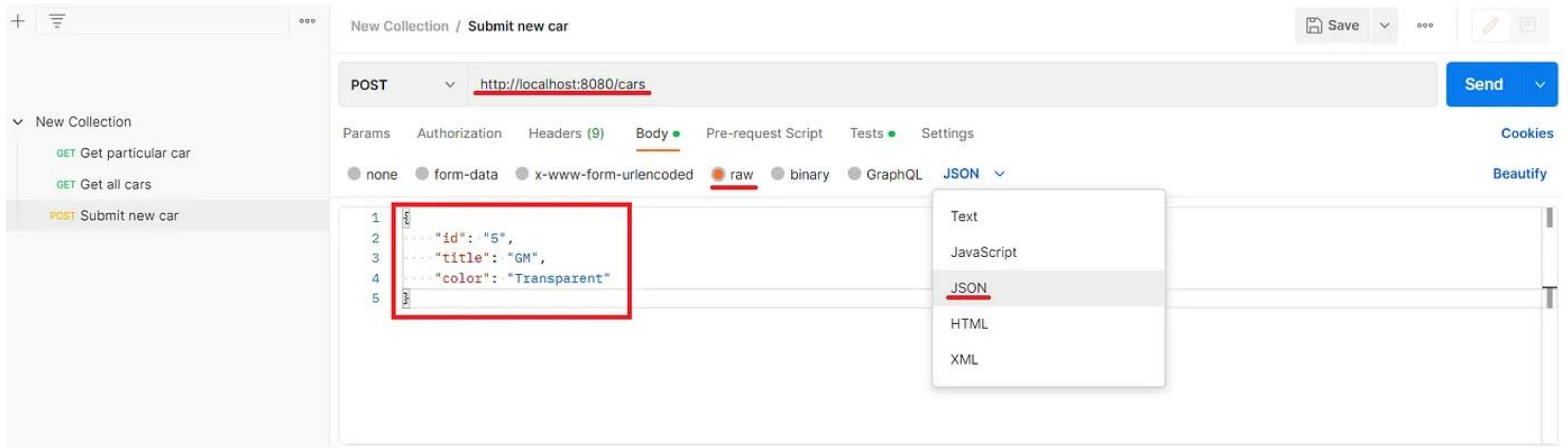
Exemplo de POST

- Para começar, clique em "Adicionar requisição".
- Selecione o método POST e adicione um ponto de extremidade com cabeçalho "Content-Type".
- Lembre-se de que ao realizar uma requisição POST, é necessário incluir um corpo (body).
- O corpo de uma requisição POST no Postman é a informação enviada.
- Para configurar o corpo da requisição, vá para a seção "**Body**".
- Escolha a opção "**Raw**" e selecione "**JSON**".
- Cole o seu conteúdo (**payload**) na área designada.

```
{  
  "id": "5",  
  "title": "GM",  
  "color": "Transparent"  
}
```

Adcionando Request

Exemplo de POST



Adicionando Testes

- Os testes no Postman são scripts usados para validar a resposta de uma API e garantir que ela funcione como esperado.
- O Postman suporta várias linguagens de script para escrever testes.
- Para usá-los, você pode adicionar scripts na seção de Snippets.
- Basta clicar em um script predefinido e ele aparecerá na sua seção de Testes.



Adcionando Testes

The screenshot shows the Postman interface for a new collection named "Submit new car". The request is a POST to `http://localhost:8080/cars`. The "Tests" tab is active, showing two test scripts:

```
1 pm.test("Status code is 201", function () {  
2   pm.response.to.have.status(201);  
3 });  
4  
5 pm.test("Body matches string", function () {  
6   pm.expect(pm.response.text()).to.include("GM");  
7 });
```

Red boxes highlight the test scripts, and red arrows point from them to the "Test Results" section on the right. The "Test Results" section shows two passed tests:

- PASS** Status code is 201
- PASS** Body matches string

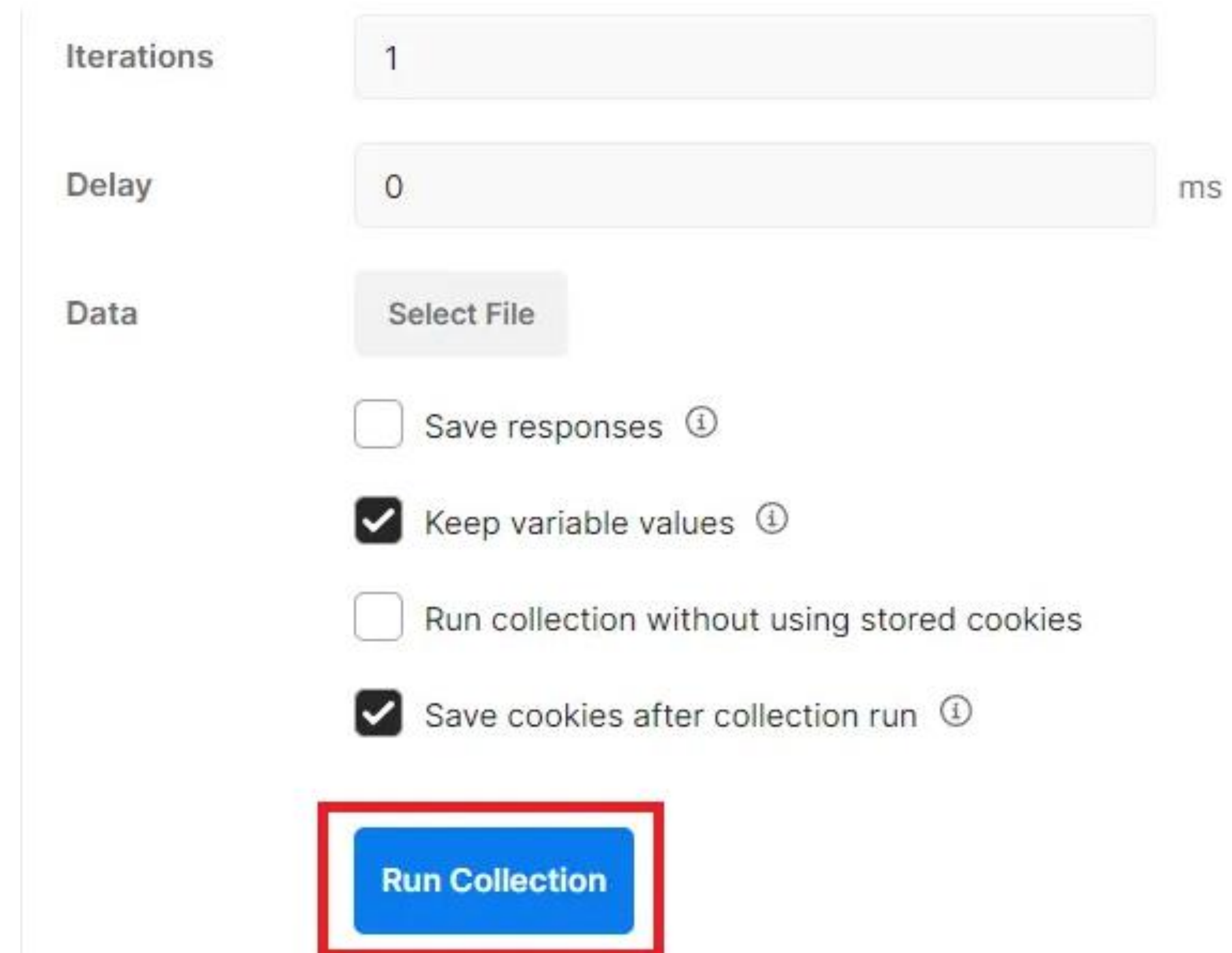
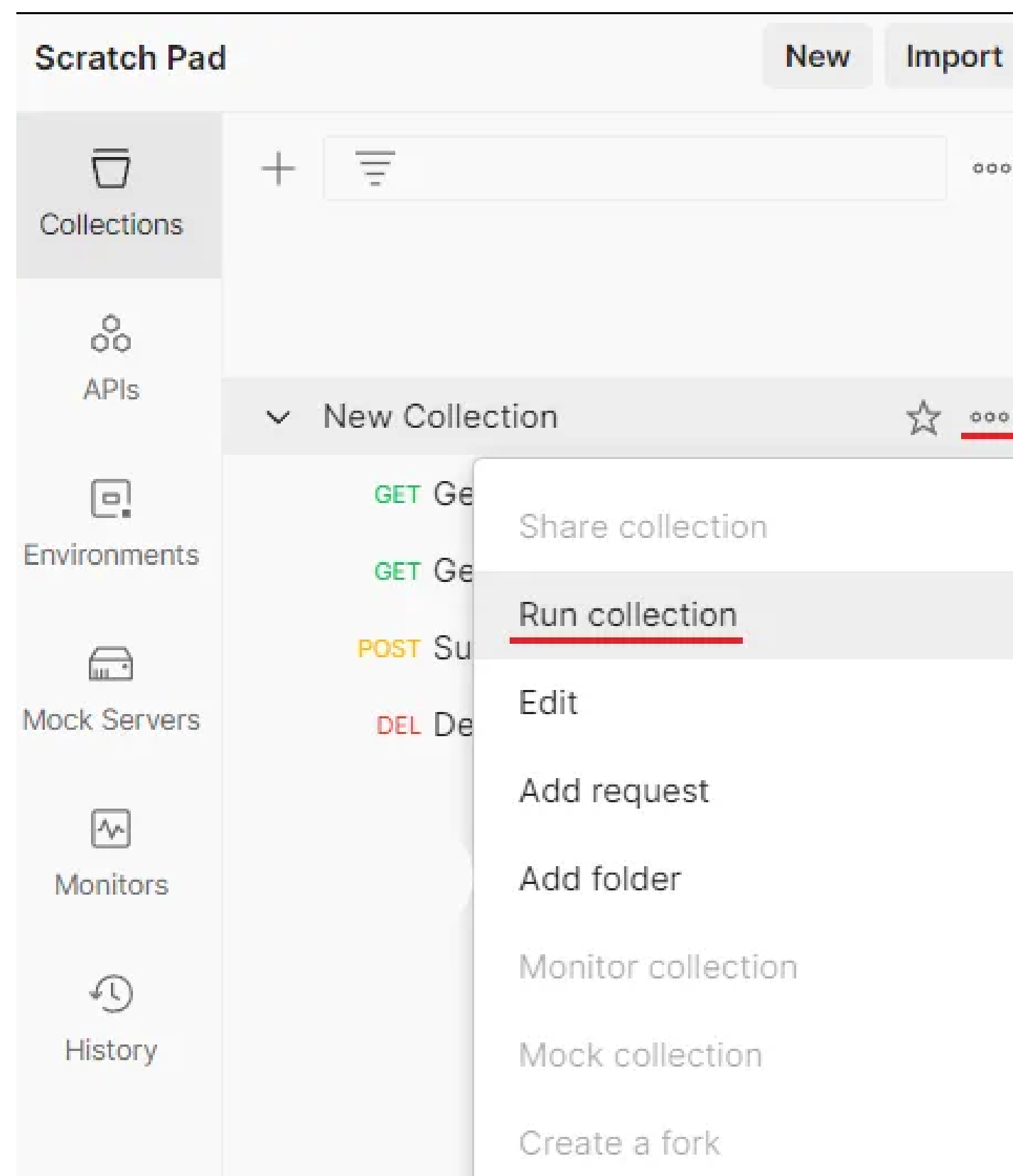
The "Test Results" section also displays the status: 201 Created, Time: 5 ms, Size: 192 B, and a "Save Response" button.

Executar sua coleção

- Como você já criou uma coleção e adicionou vários pedidos com testes a eles, você pode executar a coleção.
- Para isso, clique no botão de três pontos no lado direito do nome da sua coleção.
- Selecione "Executar coleção".

Executar sua coleção

Opções de configuração disponíveis



A large, stylized pink star with five points, positioned on the left side of the slide. It is partially obscured by the text.

Agradecemos a sua atenção!