

CAVALEIRO SPRINGBOOT



Desbravando os Mistérios
do CRUD na Era Java

VICTOR PINHEIRO

Java com Spring Boot

Simplificando o desenvolvimento com Spring Boot

Se você está buscando iniciar sua jornada no desenvolvimento de aplicativos Java de forma rápida e eficiente, o Spring Boot é uma escolha excelente. Spring Boot é um framework Java que facilita a criação de aplicativos robustos e escaláveis, fornecendo uma estrutura poderosa e repleta de funcionalidades, enquanto reduz a complexidade de configuração.

Neste Ebook, vamos guiá-lo desde os primeiros passos até a implementação de um CRUD simples utilizando o Spring Boot. Vamos aprender a configurar o ambiente de desenvolvimento, criar um novo projeto Spring Boot, definir entidades de dados, implementar operações de banco de dados, expor funcionalidades através de um controlador REST e, por fim, testar nossa aplicação para garantir que tudo esteja funcionando conforme o esperado.

Ao final deste guia, você estará familiarizado com os fundamentos do desenvolvimento com Spring Boot e terá as habilidades necessárias para construir seus próprios aplicativos Java de forma rápida e eficaz. Então, vamos começar esta jornada emocionante juntos!

01

Configurando o Ambiente de Desenvolvimento

Neste capítulo, vamos configurar o ambiente de desenvolvimento, garantindo que você tenha tudo o que precisa para começar a trabalhar com o Spring Boot.

Instalando o JDK

O Java Development Kit (JDK) é essencial para o desenvolvimento de aplicativos Java. Ele inclui o Java Runtime Environment (JRE), que é necessário para executar aplicativos Java, e o Java Development Kit (JDK), que contém as ferramentas e bibliotecas necessárias para desenvolver aplicativos Java.

Windows:

Você pode baixar o JDK mais recente do site da [Oracle](https://www.oracle.com/technetwork/java/javase-downloads-1344955.html), baixar o arquivo executável(.EXE) e seguir as instruções de instalação.

macOS:

O JDK pode ser instalado usando o Homebrew, um gerenciador de pacotes para macOS. Execute o seguinte comando no terminal:

```
Caveleiro Spring Boot - Victor Pinheiro  
brew install --cask adoptopenjdk
```

Linux (Ubuntu):

No Ubuntu, você pode instalar o JDK usando o apt, o gerenciador de pacotes padrão. Execute os seguintes comandos no terminal:

```
Caveleiro Spring Boot - Victor Pinheiro  
sudo apt update  
sudo apt install default-jdk
```


Instalando o JDK

Após a instalação do JDK, você pode verificar se tudo foi instalado corretamente digitando os seguintes comandos no terminal ou prompt de comando:



Caveleiro Spring Boot - Victor Pinheiro

```
java -version
```



Caveleiro Spring Boot - Victor Pinheiro

```
javac -version
```

Isso deve exibir a versão do JDK instalada em sua máquina.

Configurando a Variável de Ambiente

Depois de instalar o JDK, é importante configurar a variável de ambiente `JAVA_HOME` para apontar para o diretório de instalação do JDK.

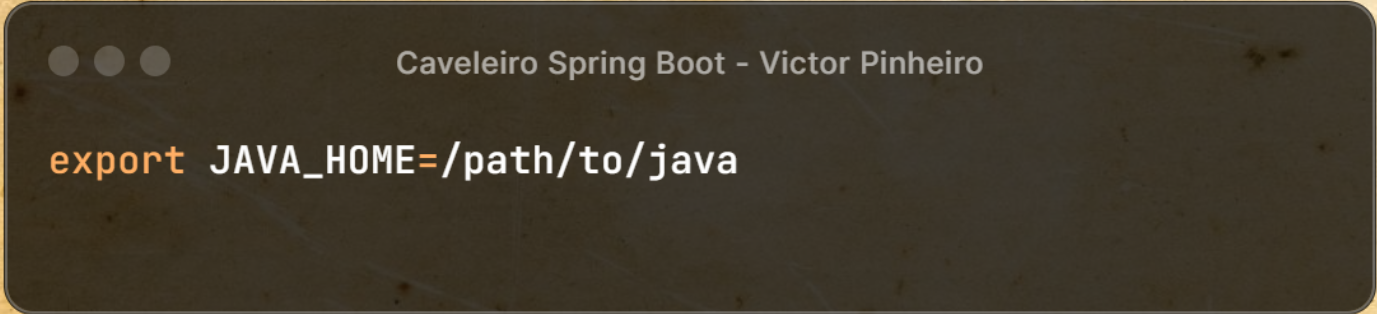
Windows:

No Windows, você pode fazer isso acessando as configurações avançadas do sistema (na barra de pesquisa do Windows, pesquise por “configurações”, após isto, no canto superior esquerdo irá encontrar “configurações avançadas do sistema”) e adicionando uma nova variável de ambiente com o nome `JAVA_HOME` e o valor do diretório de instalação do JDK (`C:\Program Files\Java`).

Dentro desse diretório, você deve encontrar uma pasta com o nome do JDK que você instalou, por exemplo, `jdk-15.0.2`.

macOS e Linux:

No macOS e no Linux, você pode adicionar a variável de ambiente `JAVA_HOME` ao arquivo de perfil do usuário, como `.bashrc` ou `.bash_profile`, adicionando a seguinte linha:



```
Caveleiro Spring Boot - Victor Pinheiro  
export JAVA_HOME=/path/to/java
```

Substitua `/path/to/java` pelo caminho do diretório de instalação do JDK em sua máquina.

Certifique-se de reiniciar seu terminal ou prompt de comando após fazer essas alterações para que as alterações entrem em vigor.

02

Criando um Novo Projeto Spring Boot

Neste capítulo, vamos criar um novo projeto Spring Boot usando o Spring Initializr e importá-lo em nossa IDE.

Gerando um Projeto com o Spring Initializr

O Spring Initializr é uma ferramenta online que nos permite gerar rapidamente um novo projeto Spring Boot com as dependências desejadas. Vamos acessar o Spring Initializr em <https://start.spring.io/> e seguir as etapas abaixo:

1. Escolha o tipo de projeto como "Maven Project" podes escolher outro se desejar.
2. Defina a linguagem de programação como "Java".
3. Selecione a versão do Spring Boot desejada.
4. Adicione a dependência "Spring Boot DevTools" ao projeto.
5. Clique no botão "Generate" para baixar o arquivo ZIP do projeto gerado.

Em seguida, vamos importar o projeto baixado em nossa IDE favorita, como IntelliJ IDEA/Eclipse.

Gerando um Projeto com o Spring Inicializr no VS Code

Para começarmos vamos instalar no nosso VS Code as extensão “Spring Boot Extension Pack”, após instalada, vamos seguir os seguintes passos:

1. Clique na barra de Search do VS Code.
2. >Spring Initializr: Create Maven Project...(no nosso caso)
3. Escolha a versão do Spring Boot
4. A linguagem JAVA
5. Nome do diretório (ex: com.demonstration)
6. Nome do projeto (ex: teste)
7. Qual packing .JAR
8. Java Version (Escolha a versão baixada no site)
9. Escolha as dependências (neste exemplo vamos utilizar: Spring Web e Spring Boot DevTools.
10. Escolha o diretório.

03

Configurações necessárias

Neste capítulo, vamos configurar nosso projeto para criar tabelas de forma automática, com as configurações de propriedades.

Configurando as dependências

Antes de tudo vamos adicionar no nosso arquivo pom.xml entre as dependências a dependência do Jakarta.

Desta forma:

```
Caveleiro Spring Boot - Victor Pinheiro

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.33</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>jakarta.persistence</groupId>
  <artifactId>jakarta.persistence-api</artifactId>
  <version>3.1.0</version>
</dependency>
```


Configurando as propriedades

E também vamos editar nossa application.properties que vem por padrão no projeto, para application.yml, pois fica mais simples de manipular.

Obs: Troque o ddl-auto para update após rodar a aplicação, senão ira criar sempre um novo banco de dados a cada inicio da aplicação

Adicionaremos nossas configurações:

```
Caveleiro Spring Boot - Victor Pinheiro

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ebookdb
    username: root
    password:
  jpa:
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect
      show-sql: true
```

Troque o username e o password para o do seu banco de dados, se tiver dificuldade para criar banco de dados, veja a seção bônus.

04

Criando uma entidade JPA

Neste capítulo, vamos criar uma entidade e implementar um repositório para manipular operações de banco de dados relacionadas à nossa entidade.

Criando uma entidade JPA

Vamos criar uma entidade para interagir com o banco de dados e executar operações CRUD em nossos objetos.

Crie um novo diretório com o nome de Model dentro do seu diretório principal (main/seu-repo) e então crie sua entidade Java.

Observe as anotações que ficam acima dos atributos e da classe(`@NomeDaAnnotation`) que vão indicar o que será feita para nossas dependências.

A annotation `@Table` tem um papel muito importante pois ela irá sinalizar qual será o nome da tabela a ser criada na nossa database, portanto é importante não esquecer de passar este parâmetro com o nome desejado para a criação da tabela, neste nosso caso `@Table(name = "product")`

Exemplo de entidade

```
Caveleiro Spring Boot - Victor Pinheiro

package com.seupackage.aqui

import java.io.Serializable;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "product")
public class Product implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "price")
    private double price;

    //Getters and Setters
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
}
```


Criando um Repositório JPA

Vamos criar um repositório JPA para interagir com o banco de dados e executar operações CRUD em nossos objetos.

Crie um novo diretório, com o nome “repositories” e dentro dele crie seu repositório JPA, normalmente seguimos uma convenção para nomes de repositórios passando a palavra “Repository” após o nome do seu repositório.

Ex: ProductRepository.java

```
Caveleiro Spring Boot - Victor Pinheiro

import org.springframework.data.jpa.repository.JpaRepository;
import com.demonstration.testando.model.Product;

public interface ProductRepository extends JpaRepository<Product, Long> {
}
```


05

Desenvolvendo um Controller REST

Neste capítulo, vamos criar um controlador REST para expor as funcionalidades relacionadas à nossa entidade.

Criando um Controller

Vamos acessar o diretório

“src\main\java\com\NomeDoSeuDiretorio\NomeDoProjeto”,
neste exemplo, ficou:

“src\main\java\com\demonstration\testando”

Onde vamos criar uma nova pasta chamada **Controller**,
nesta pasta é onde vamos criar um controller para lidar com
as solicitações HTTP. Por exemplo:

```
Caveleiro Spring Boot - Victor Pinheiro

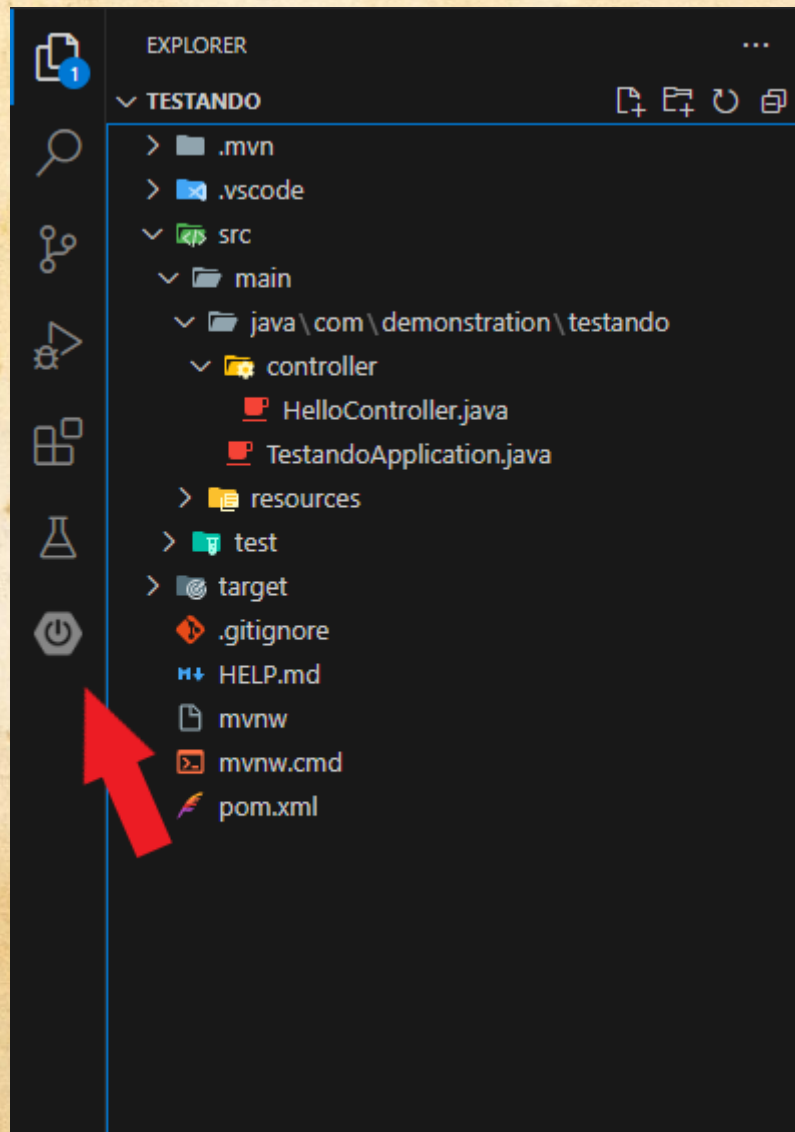
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

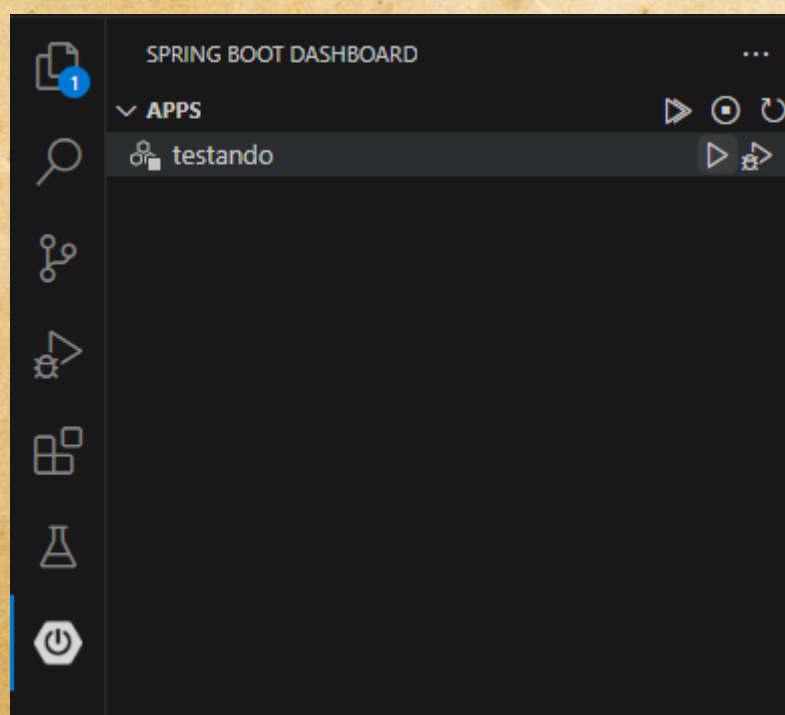
    @GetMapping("/hello")
    public String helloWorld() {
        return "Hello, world!";
    }
}
```


Executando o Aplicativo

No canto inferior esquerdo da barra de tarefas do VS code, podemos encontrar o Spring Boot Dashboard:



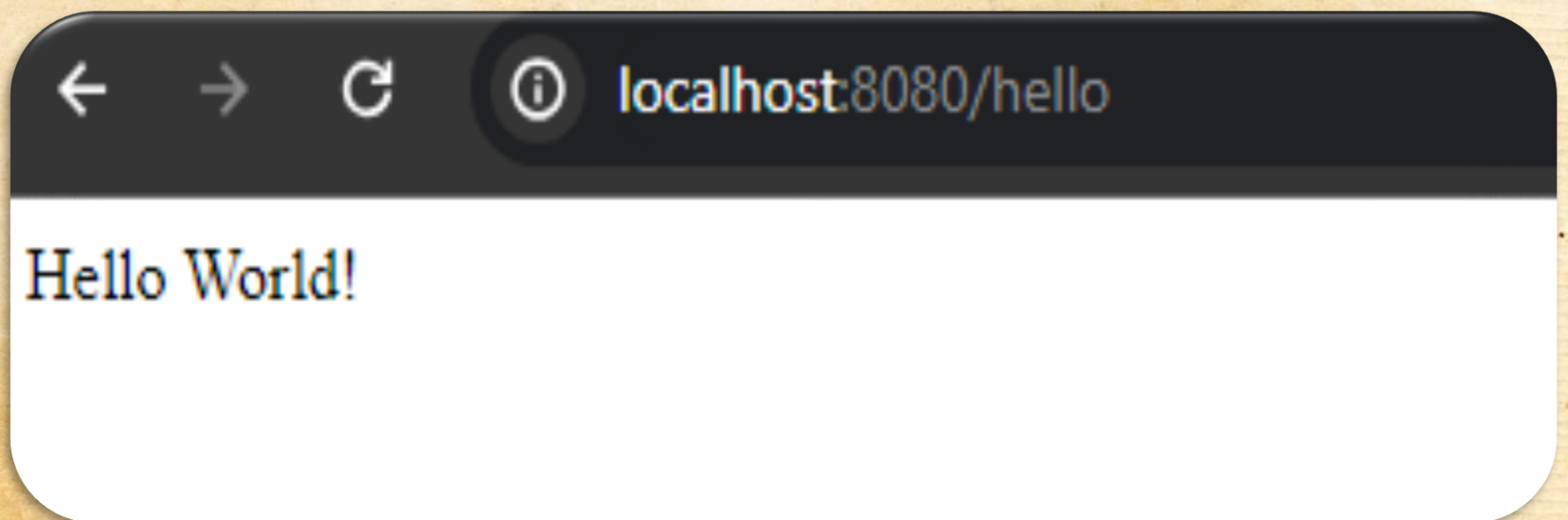
Vamos dar start(run) na nossa aplicação Spring Boot:



Executando o Aplicativo

Execute o aplicativo Spring Boot no VS Code e acesse a URL "http://localhost:8080/hello" em seu navegador para ver a mensagem "Hello, world!".

Pronto, isso mostra que seu aplicativo esta funcionando até aqui!



Criando um Rest Controller

Depois de ter testado seu aplicativo a funcionar, vamos agora implementar um CRUD com o controller utilizando aquela entidade (na pasta Model) que criamos, manipular as requisições HTTP relacionadas aos produtos.

Vamos fazer o CREATE e o READ do nosso CRUD:

```
Caveleiro Spring Boot - Victor Pinheiro

@RestController
@RequestMapping("/products/")
public class ProductController {

    @Autowired
    ProductRepository repository;

    @PostMapping("/add")
    public Product createProduct(@RequestBody Product product) {
        var pdt = repository.save(product);
        return pdt;
    }

    @GetMapping
    public List<Product> getAll() {

        List<Product> all = repository.findAll();
        return all;
    }
}
```


Criando um Rest Controller

Vamos fazer o UPDATE e o DELETE do nosso CRUD:

```
Caveleiro Spring Boot - Victor Pinheiro

@PutMapping("update/{id}")
public Product updateProduct(@PathVariable Long id, @RequestBody Product
product) {
    Product upProduct = repository.findById(id).orElseThrow();

    product.setId(id);
    upProduct.setName(product.getName());
    upProduct.setPrice(product.getPrice());

    repository.save(product);
    return product;
}

@DeleteMapping("delete/{id}")
public String deleteProduct(@PathVariable Long id){
    Product getProduct = repository.findById(id).orElseThrow();

    repository.delete(getProduct);

    return "Produto deletado";
}
}
```

Observe a utilização das Annotations para indicar qual será o tipo de requisição HTTP a ser feita.

06

Testando a Aplicação

Neste capítulo, vamos testar de forma simples nossa aplicação utilizando o Postman.

O poderoso Postman

De forma geral, teríamos que criar um Frontend para nossa aplicação para testarmos nossa aplicação e seria muito trabalhoso apenas para estes fins.

O Postman é uma ferramenta muito útil para testar APIs RESTful, como as que criamos em nosso projeto Spring Boot. Ele fornece uma interface gráfica amigável para enviar solicitações HTTP para seus endpoints e visualizar as respostas.

Vamos começar testando o nosso CREATE, para isso precisamos passar em forma de JSON nossos atributos que criamos na nossa entidade, neste caso “Name” e “Price”, o ID será criado de forma automática por conta da Annotation que colocamos.

Logo em seguida vamos testar nosso GET e observar se os dados foram armazenados corretamente.

Após, utilizaremos o PUT, para fazer o update do item adicionado e observar se foram alterados seus valores.

Por fim, vamos deletar o item, que retornará uma mensagem de sucesso se o item for mesmo deletado.

0 poderoso Postman

CREATE(POST):

The screenshot shows the Postman interface for a POST request. The URL bar at the top displays 'http://localhost:8080/products/add'. Below the URL bar, the 'Body' tab is selected, and the 'JSON' format is chosen. The request body contains a JSON object with 'name' and 'price' fields. The response section at the bottom shows a 200 OK status with a JSON body containing 'id', 'name', and 'price' fields.

```
GET http://localhost:8080/products/add
```

Send

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

none form-data x-www-form-urlencoded raw binary **JSON**

```
1 {
2   "name": "Smartphone",
3   "price": 2000
4 }
```

Body Cookies Headers (5) Test Results 200 OK 358 ms 207 B Save Response

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": 1,
3   "name": "Smartphone",
4   "price": 2000.0
5 }
```

READ(GET):

The screenshot shows the Postman interface for a GET request. The URL bar at the top displays 'http://localhost:8080/products/'. Below the URL bar, the 'Body' tab is selected, and the 'none' format is chosen. The response section at the bottom shows a 200 OK status with a JSON body containing 'id', 'name', and 'price' fields.

```
GET http://localhost:8080/products/
```

Send

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

none form-data x-www-form-urlencoded raw binary

This request does not have a body

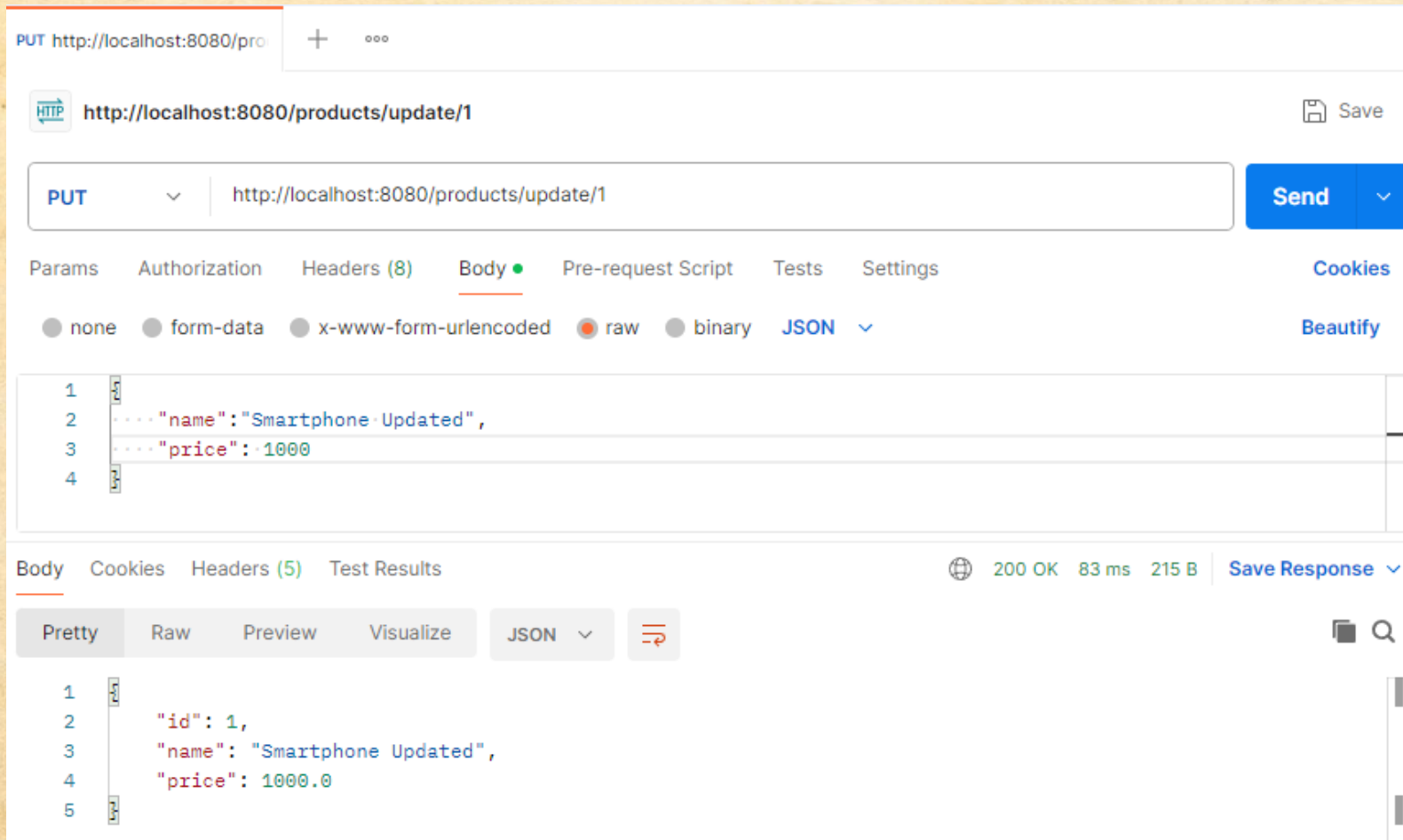
Body Cookies Headers (5) Test Results 200 OK 26 ms 209 B Save Response

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": 1,
3   "name": "Smartphone",
4   "price": 2000.0
5 }
```


0 poderoso Postman

UPDATE(PUT):



The image shows the Postman interface for a PUT request. The URL bar at the top shows 'PUT http://localhost:8080/products/update/1'. Below the URL bar, the 'Body' tab is selected, and the request body is a JSON object:

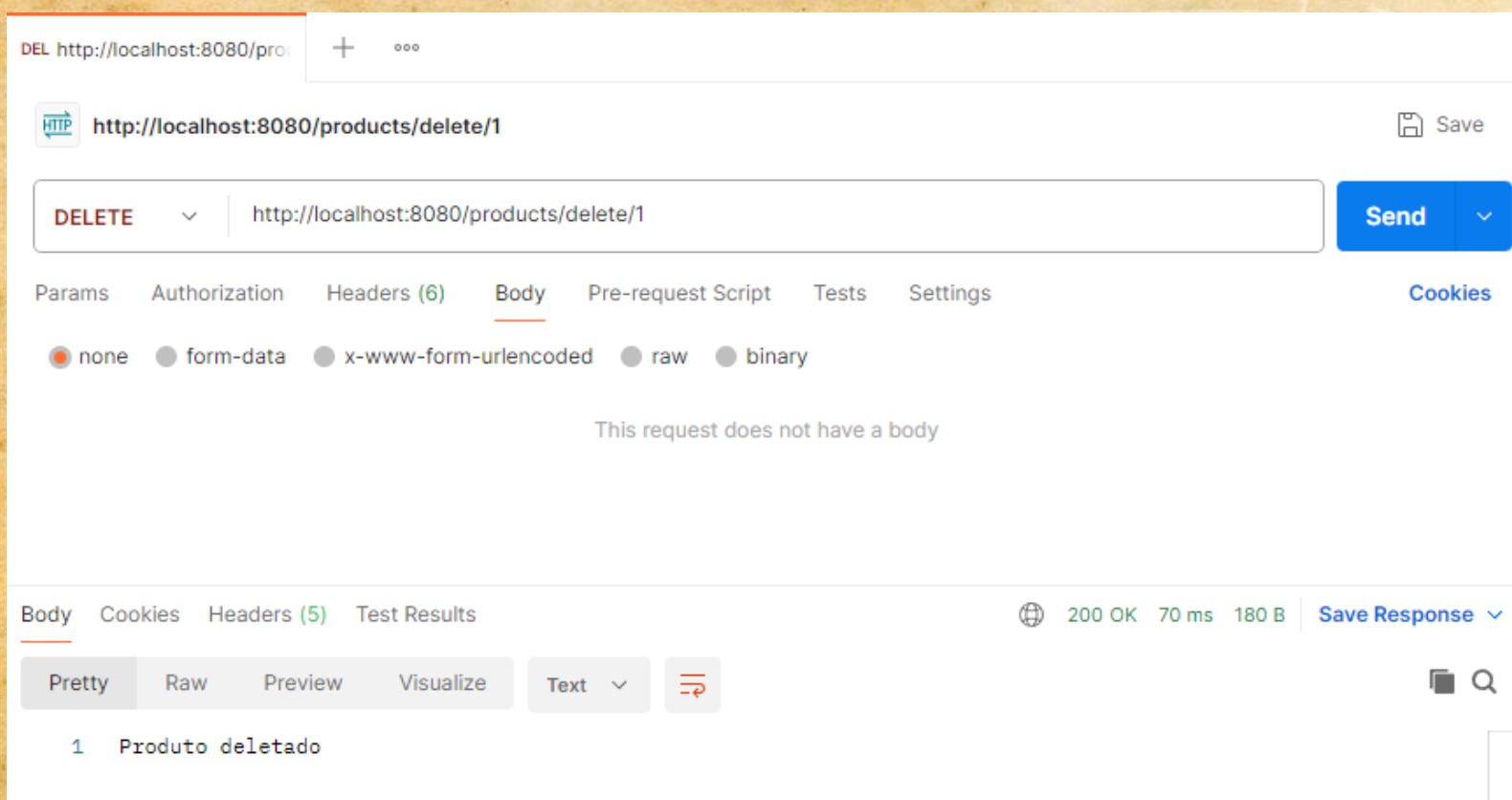
```
{  "name": "Smartphone Updated",  "price": 1000}
```

. The 'Send' button is visible. Below the request, the 'Response' tab is selected, showing a 200 OK status with a response body:

```
{  "id": 1,  "name": "Smartphone Updated",  "price": 1000.0}
```

. The interface includes tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings. The 'Body' tab is currently active, and the 'JSON' format is selected.

DELETE:



The image shows the Postman interface for a DELETE request. The URL bar at the top shows 'DEL http://localhost:8080/products/delete/1'. Below the URL bar, the 'Body' tab is selected, and the message 'This request does not have a body' is displayed. The 'Send' button is visible. Below the request, the 'Response' tab is selected, showing a 200 OK status with a response body:

```
1  Produto deletado
```

. The interface includes tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. The 'Body' tab is currently active, and the 'Text' format is selected.

Bônus

Configurando o MySQL server do zero.

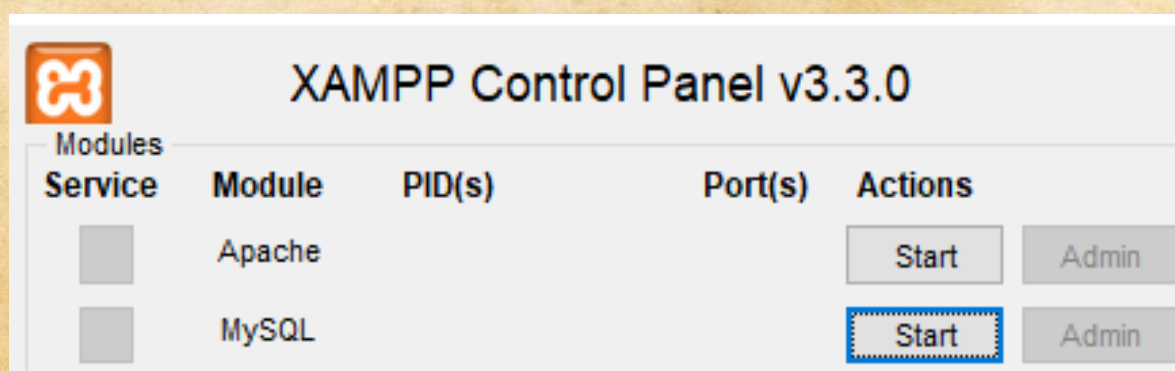
Neste capítulo bônus, se você não teve nenhuma experiência previa de como fazer uma conexão com o MySQL vou te ensinar de forma simples como começar para fazer seus testes.

Iniciando as configurações

Vamos começar fazendo o download do Xampp (https://www.apachefriends.org/pt_br/download.html) e instalando-o na sua maquina.

Vamos fazer download do MySQL Workbench(<https://dev.mysql.com/downloads/workbench/>)e instala-lo para observarmos nossa tabela criada e assim vendo se cada ação do CRUD está agindo corretamente.

Inicie o XAMPP e de start no MySQL



Abra o MySQL Workbench, observe que no MySQL Connections terá uma connection padrão sem senha, criada pelo XAMPP.

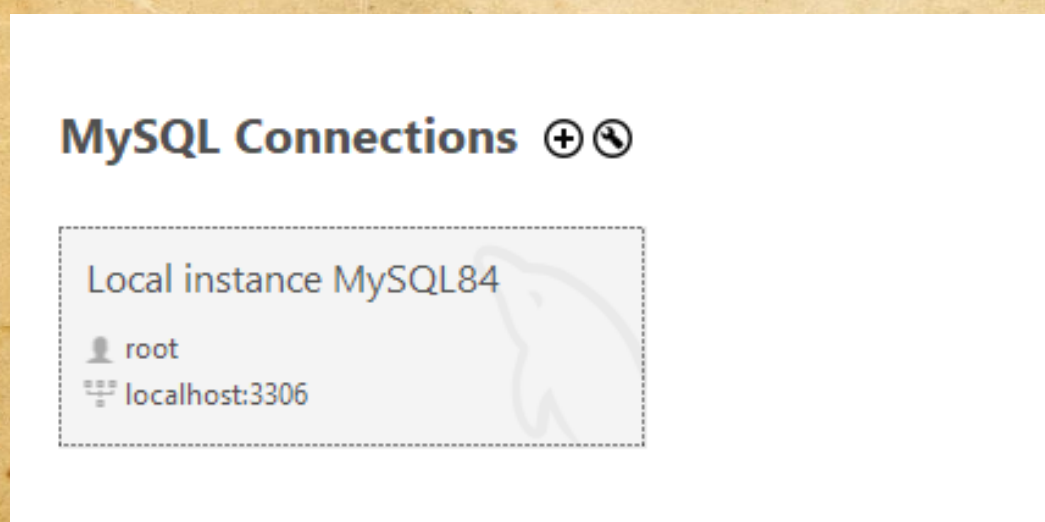
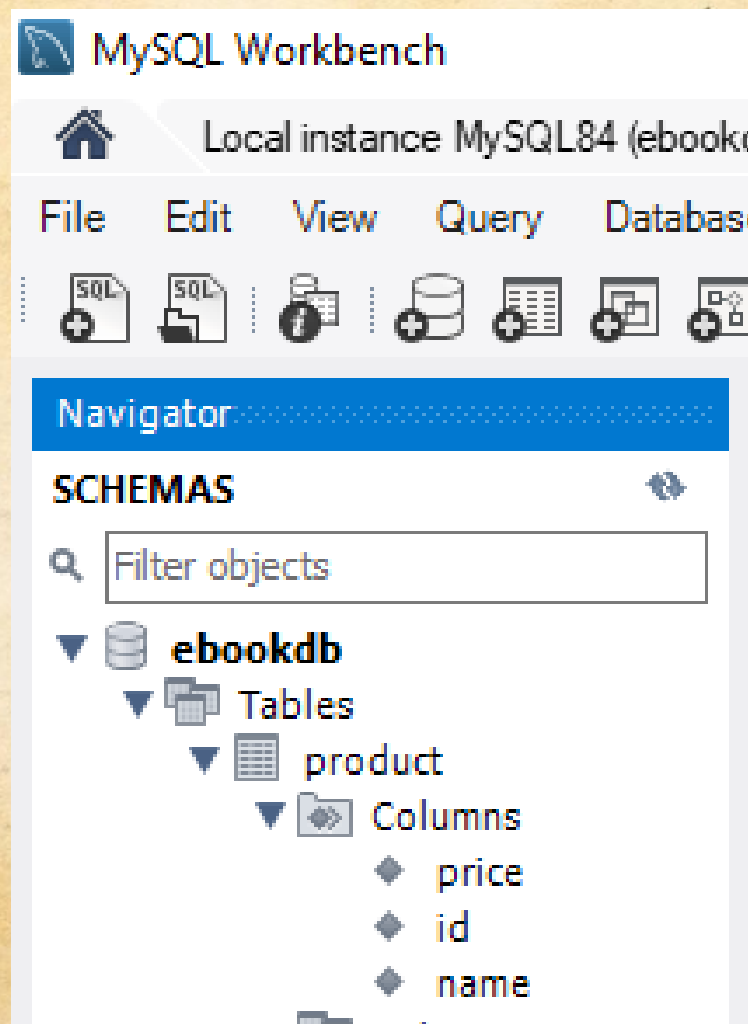
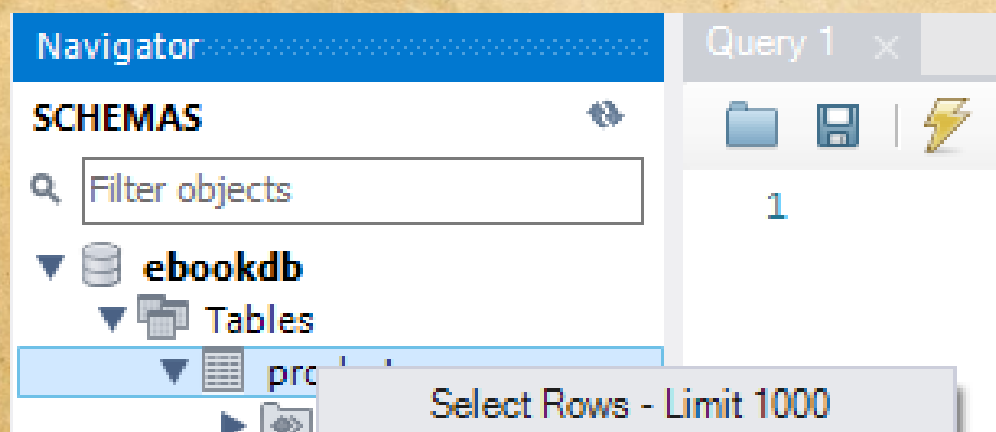


Table e Rows

Agora observe que temos nosso bando de dados com nossa tabela (se já tiver seguido os passos anteriores).



Observe as colunas da tabela com os valores, clicando com o botão direito em cima da sua tabela e clicando em “Select Rows”:



Finalizando

Conclusão

Neste ebook, você deu os primeiros passos para criar um aplicativo robusto utilizando Java Spring Boot e MySQL. Desde a configuração do ambiente de desenvolvimento até a implementação de um CRUD simples, você aprendeu os fundamentos essenciais para iniciar seu projeto.

No entanto, este é apenas o ponto de partida. O universo do desenvolvimento com Spring Boot é vasto e repleto de possibilidades. À medida que você avança em sua jornada, pode explorar recursos avançados, como segurança, testes automatizados, integração com outros serviços e muito mais.

Continue praticando e experimentando com o código fornecido neste ebook. Desafie-se a expandir suas habilidades e a criar aplicativos cada vez mais sofisticados. E lembre-se, a comunidade Spring Boot é vasta e está sempre pronta para ajudar. Não hesite em buscar suporte online, participar de fóruns e contribuir para projetos open-source.

Com dedicação e persistência, você será capaz de construir aplicativos poderosos e inovadores com Java Spring Boot. Boa sorte em sua jornada de desenvolvimento e que seus projetos alcancem todo o sucesso que você merece!

Obrigado Por ler até aqui

Aos valorosos leitores de "O Cavaleiro SpringBoot",
Saudações e profundo agradecimento por embarcarem nesta jornada através das páginas de nosso tomo sagrado. Nesta obra, buscou-se oferecer uma introdução concisa e prática para que possais empunhar vossas ferramentas e forjar vossas próprias APIs utilizando Java e o poderoso artefato conhecido como Spring Boot.

Sabei, destemidos aventureiros, que a prática incessante é a chave para alcançar a maestria. Espero que este compêndio tenha instigado vossa curiosidade e determinação para continuar vossos estudos e práticas sempre que possível. Cada linha de código escrita, cada erro desvendado, e cada funcionalidade conquistada é uma vitória em vossa honrada jornada de conhecimento e habilidade.

Lembraí-vos sempre de que o aprendizado é um caminho sem fim. O vasto reino da tecnologia está em constante transformação, e sempre há novas maravilhas a serem descobertas e dominadas. Continuai explorando, questionando e, sobretudo, codificando. A dedicação e esforço que despendes agora certamente trarão grandes recompensas em vossas futuras aventuras.

Agradeço-vos, nobres leitores, por depositardes vossa confiança em "O Cavaleiro SpringBoot" como guia em vossa jornada de desenvolvimento. Que vossos empreendimentos tecnológicos sejam coroados com sucesso e que encontreis grande satisfação em vossas conquistas.

Com eterna gratidão,
Victor Pinheiro

