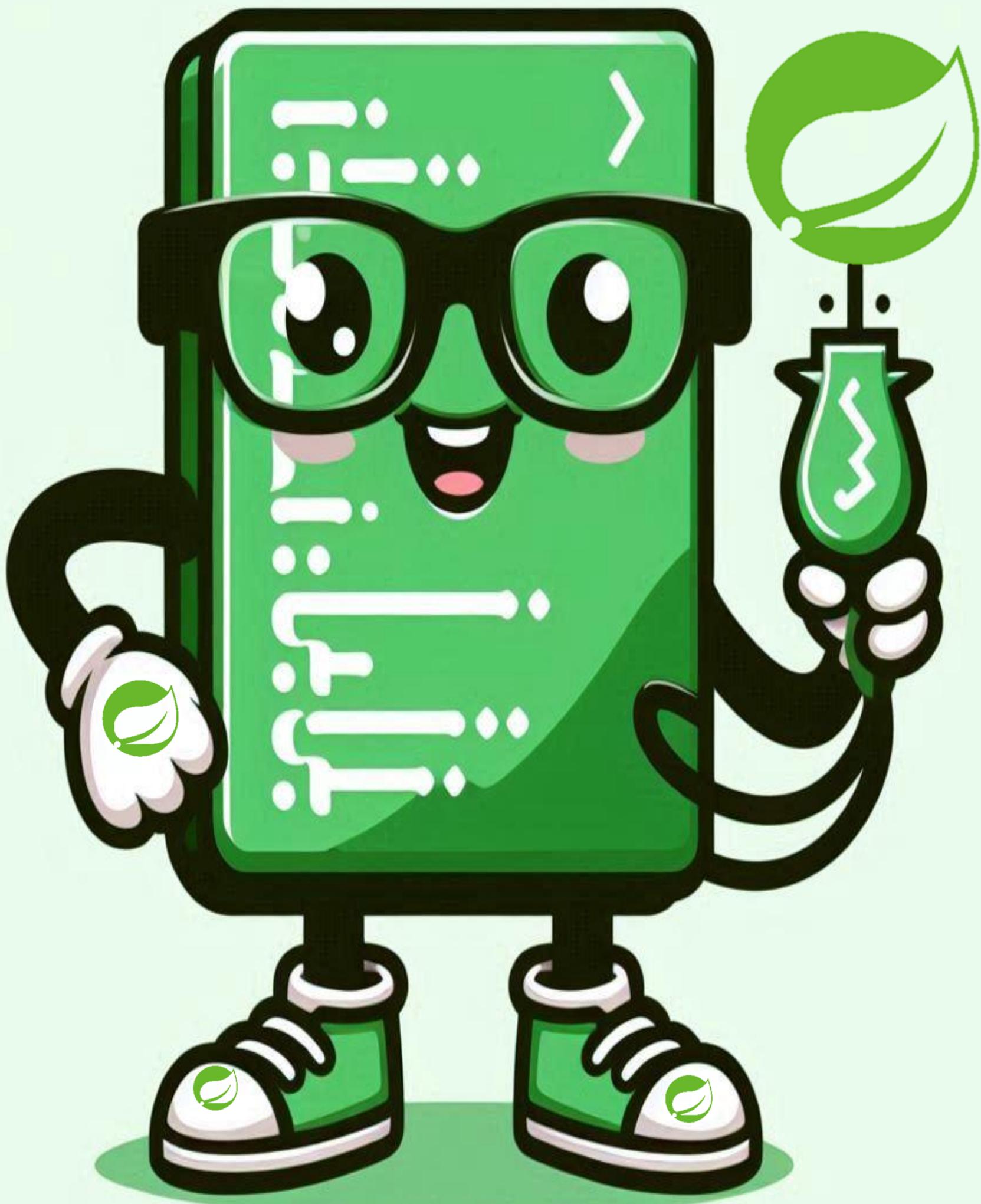


SPRING BOOT



O DESPERTAR DA PROGRAMAÇÃO

RAFAEL RAMOS MACHADO

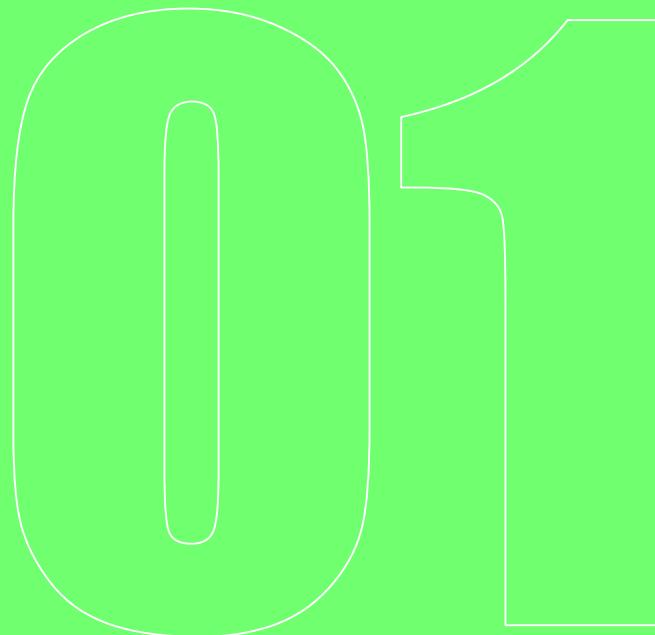
APRENDENDO SPRING BOOT DE FORMA PRÁTICA

Criando um CRUD Básico com Spring Boot

Spring Boot é uma ferramenta poderosa que simplifica o desenvolvimento de aplicações Java, especialmente para a criação de APIs RESTful.

Neste guia, vamos mostrar como criar um CRUD (Create, Read, Update, Delete) básico usando Spring Boot.





CONFIGURAÇÃO DO PROJETO



INICIANDO UM NOVO PROJETO SPRING BOOT

Para começar, crie um novo projeto Spring Boot usando o Spring Initializr (<https://start.spring.io/>).

Selecione as seguintes dependências:

Spring Web

Spring Data JPA

H2 Database (ou outra de sua preferência)

The screenshot shows the Spring Initializr web application interface. At the top, there's a browser header with tabs and icons. Below it is the Spring Initializr logo. The main form has sections for Project, Language, and Project Metadata, followed by a large table listing selected dependencies.

Project		Language			Dependencies		
<input type="radio"/> Gradle - Groovy	<input type="radio"/> Gradle - Kotlin	<input checked="" type="radio"/> Java	<input type="radio"/> Kotlin	<input type="radio"/> Groovy	ADD DEPENDENCIES... CTRL + B		
<input checked="" type="radio"/> Maven							
Spring Boot							
<input type="radio"/> 3.4.0 (SNAPSHOT)	<input type="radio"/> 3.4.0 (M1)	<input type="radio"/> 3.3.3 (SNAPSHOT)	<input checked="" type="radio"/> 3.3.2				
<input type="radio"/> 3.2.9 (SNAPSHOT)	<input type="radio"/> 3.2.8						
Project Metadata							
Group	com.example						
Artifact	teste						
Name	teste						
Description	Teste Spring Boot						
Package name	com.example.teste						
Packaging	<input checked="" type="radio"/> Jar	<input type="radio"/> War					
Java	<input type="radio"/> 22	<input checked="" type="radio"/> 21	<input type="radio"/> 17				
GENERATE CTRL + D EXPLORE CTRL + SPACE SHARE...							

ESTRUTURA BÁSICA DO PROJETO

Após gerar o projeto, você verá uma estrutura de pastas similar a esta:

```
src
└ main
  └ java
    └ com.example.demo
      └ DemoApplication.java
      └ controller
      └ model
      └ repository
    └ resources
      └ application.properties
```

09

MODELAGEM DE DADOS

CRIANDO A ENTIDADE

Vamos criar uma entidade chamada **Product** que representa um produto.

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

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;

    // Getters e Setters
}
```

03

REPOSITÓRIO

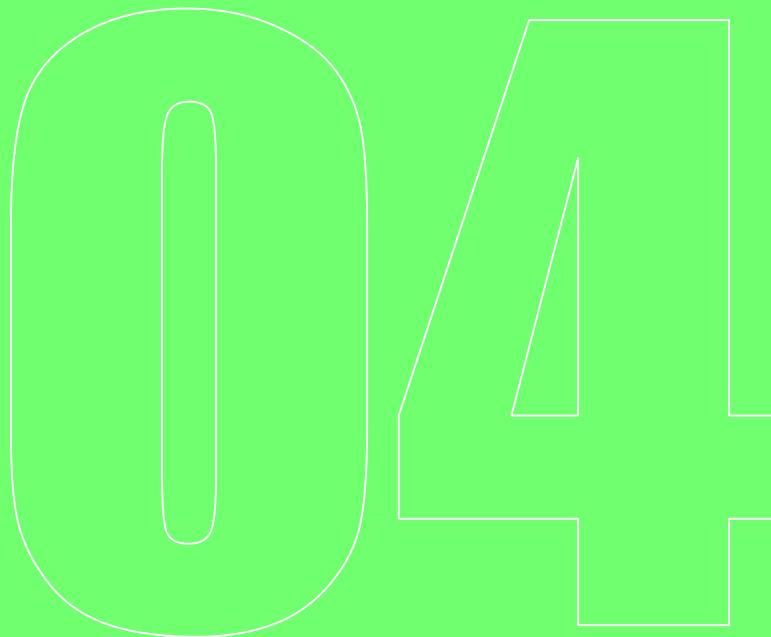
CRIANDO O REPOSITÓRIO

Agora, vamos criar um repositório para a entidade Product:

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

import org.springframework.data.jpa.repository.JpaRepository;
import com.example.demo.model.Product;

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



CONTROLADOR

CRIANDO O CONTROLADOR REST

O controlador gerencia as requisições HTTP. Vamos criar um controlador para nosso CRUD.

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import com.example.demo.model.Product;
import com.example.demo.repository.ProductRepository;

import java.util.List;

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

    @Autowired
    private ProductRepository productRepository;

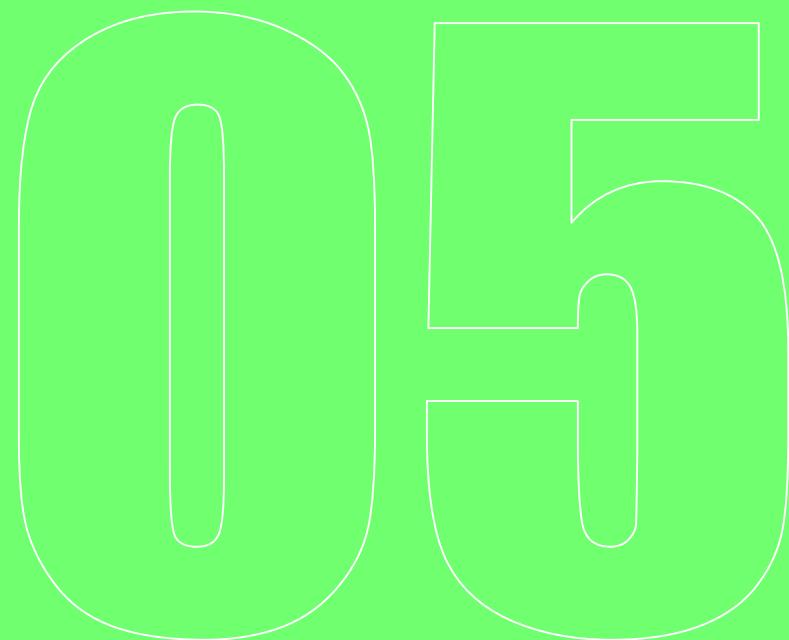
    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) {
        return productRepository.findById(id).orElse(null);
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }

    @PutMapping("/{id}")
    public Product updateProduct(@PathVariable Long id, @RequestBody Product productDetails) {
        Product product = productRepository.findById(id).orElse(null);
        if (product != null) {
            product.setName(productDetails.getName());
            product.setPrice(productDetails.getPrice());
            return productRepository.save(product);
        }
        return null;
    }

    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) {
        productRepository.deleteById(id);
    }
}
```



CONFIGURAÇÃO DO BANCO DE DADOS

CONFIGURANDO O H2 DATABASE

Adicione as seguintes propriedades no arquivo `application.properties` para configurar o banco de dados H2:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

06

EXECUTANDO A APLICAÇÃO

INICIANDO O PROJETO

Para rodar a aplicação, execute a classe `DemoApplication.java`.

Acesse o console do H2 em <http://localhost:8080/h2-console> para verificar o banco de dados.





TESTANDO A API

TESTANDO COM cURL

Aqui estão alguns exemplos de comandos cURL para testar a API:

Criar um produto:

```
curl -X POST -H "Content-Type: application/json" -d  
'{"name":"Laptop","price":1200.00}' http://localhost:8080/products
```

Obter todos os produtos:

```
curl http://localhost:8080/products
```

Obter um produto pelo ID:

```
curl http://localhost:8080/products/1
```

Atualizar um produto:

```
curl -X PUT -H "Content-Type: application/json" -d  
'{"name":"Laptop Pro","price":1500.00}' http://localhost:8080/products/1
```

Deletar um produto:

```
curl -X DELETE http://localhost:8080/products/1
```

08

CONCLUSÃO

FUNDAMENTOS BÁSICOS

Neste guia, você aprendeu a criar um CRUD básico com Spring Boot, configurando uma entidade, repositório e controlador REST. Com esses fundamentos, você pode expandir e personalizar sua aplicação conforme necessário.

