APLICAÇÕES WEB

Prof. Me. André Roberto da Silva

Introdução ao Spring e Spring Boot

Spring

O Spring Framework é um framework Java de código aberto que simplifica o desenvolvimento de aplicações robustas, resolvendo problemas como a configuração manual de objetos, acoplamento excessivo e dificuldade de testes.

Principais características:

- Inversão de Controle (IoC): o Spring gerencia os objetos e suas dependências.
- AOP: facilita a implementação de funcionalidades como segurança e logging.
- Integração fácil com tecnologias como Hibernate, JPA, JDBC.
- Módulos reutilizáveis, como Spring MVC e Spring Security.

Spring

https://spring.io/ → Página do projeto Spring

Most [of our] services today are all based on Spring Boot. I think the most important thing is that [Spring] has just been very well maintained over the years...that is important for us for the long term because we don't want to be switching to a new framework every two years.

Paul Bakker, Senior Software Engineer, Netflix

Watch now

Spring Boot

Spring Boot é um framework Java de código aberto, baseado no Spring Framework, que simplifica o processo de configuração e desenvolvimento de aplicações.

Permite criar aplicações *stand-alone* (autônomas) com o mínimo de configuração necessária, eliminando a complexidade de *setups* tradicionais e facilitando a integração com outras ferramentas e bibliotecas.

Spring Boot

Principais características:

- Configuração automática: configura automaticamente os componentes necessários da aplicação, com base nas dependências adicionadas.
- Aplicações autônomas (*stand-alone*): permite criar aplicações independentes que já incluem um servidor embutido, como o Tomcat, sem a necessidade de instalar ou configurar servidores externos.
- Pronto para produção: Fornece configurações que tornam o ambiente de produção mais seguro e otimizado, como monitoramento e métricas.
- Spring Boot Starter: fornece "starters" (dependências prontas) para integração fácil com diversos frameworks e ferramentas, como Spring Data, Spring Security, Spring MVC, entre outros.
- Minimalismo de configuração: reduz a quantidade de configuração necessária, adotando convenções para configuração automática e evitando configurações complexas no arquivo *application.properties*.

Configuração da IDE

Utilizaremos a IDE VSCode, porém se faz necessária a configuração antes da utilização com projetos Spring Boot.

Instale as duas extensões:

Extension Pack for Java

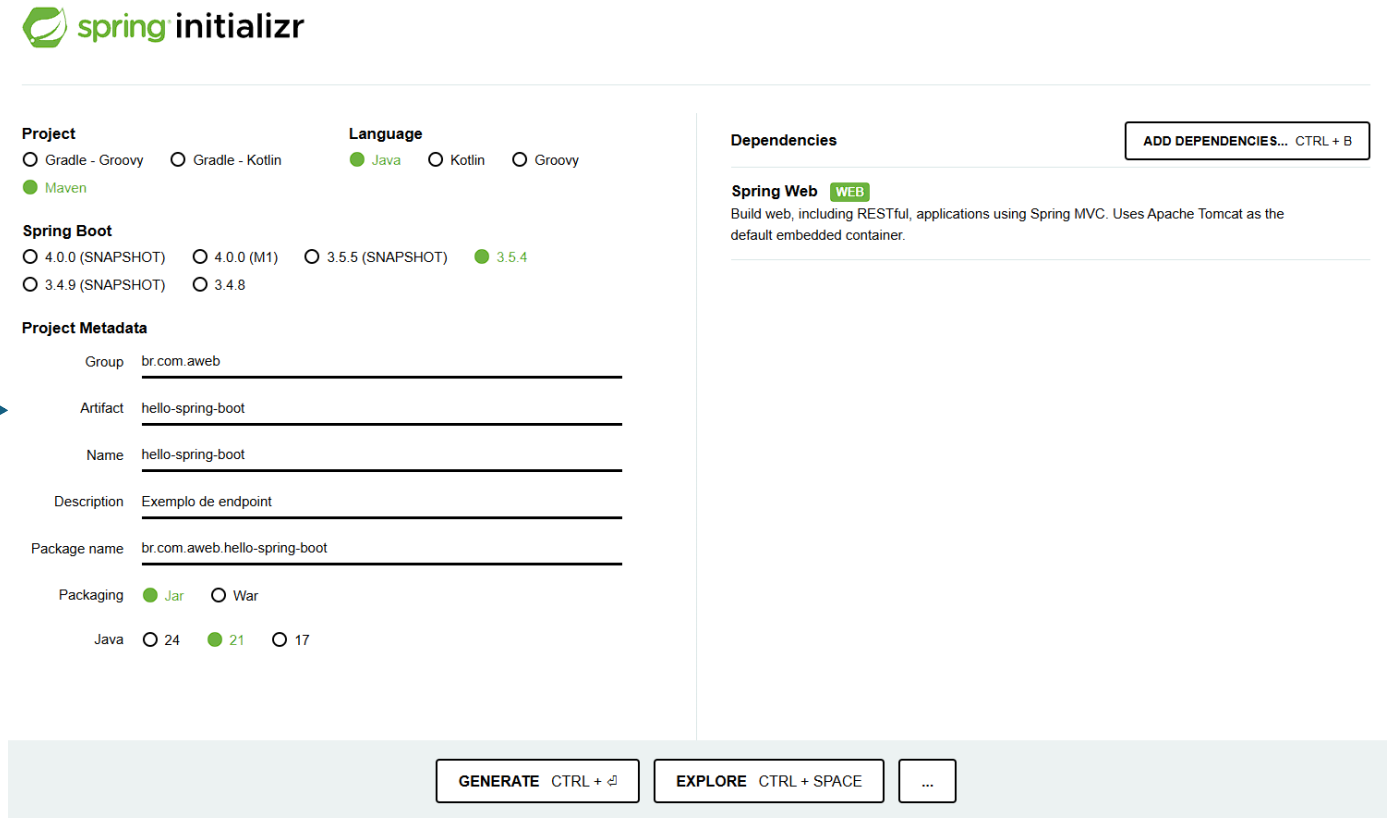
Microsoft

Spring Boot Extension Pack

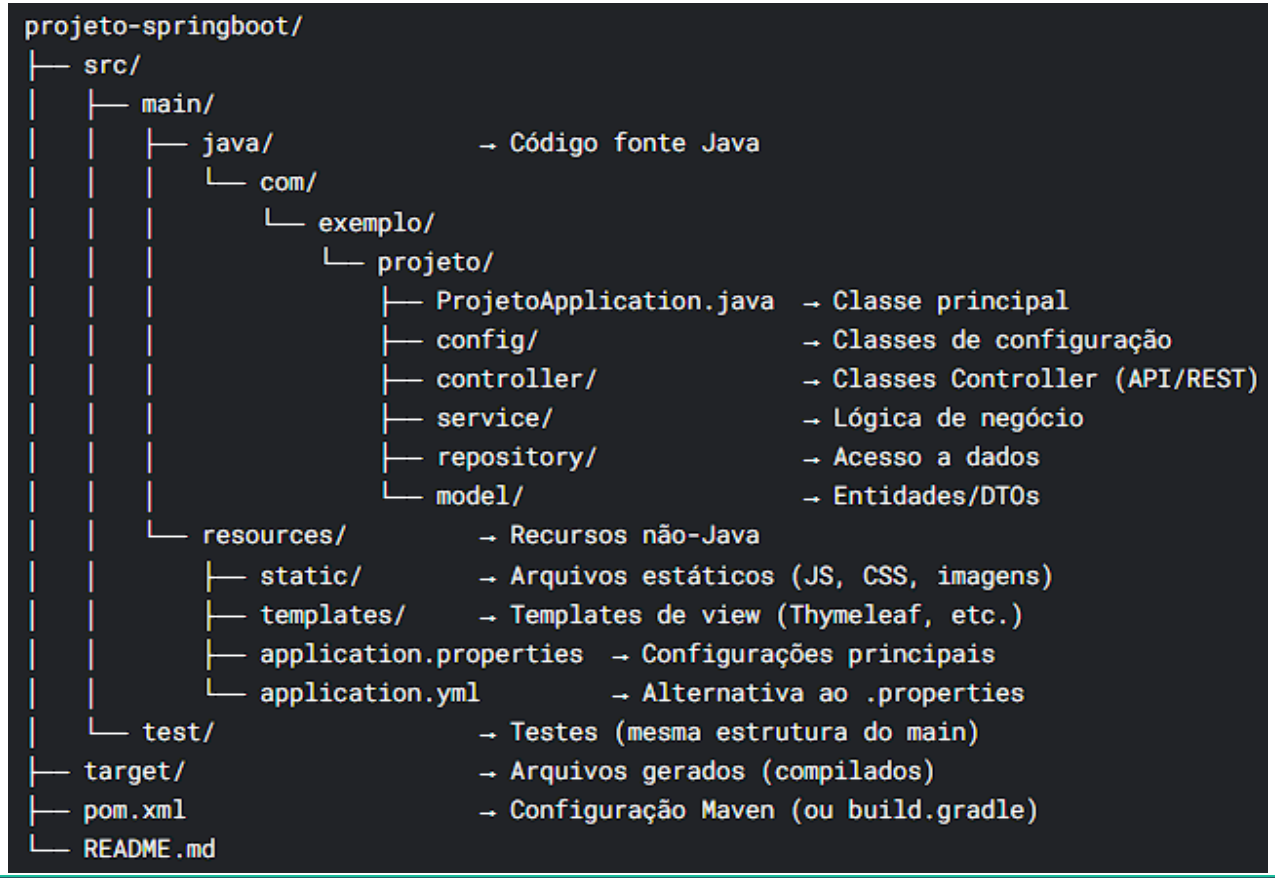
VMware

Exemplo Hello World Spring Boot

start.spring.io



Exemplo Hello World Spring Boot



Exemplo Hello World Spring Boot

Padrão MVC (Model-View-Controller)

O **MVC** é um padrão de arquitetura de software que separa uma aplicação em três componentes principais:

Model (Modelo)

Representa os dados e a lógica de negócio.

- Entidades (classes JPA/Hibernate para banco de dados)
- DTOs (objetos para transferência de dados)
- Serviços (regras de negócio)

View (Visão)

Responsável pela interface do usuário (UI).
Templates (Thymeleaf, JSP) para renderização no servidor.
Em APIs REST a "View" pode ser substituída por JSON/XML (retornado pelo Controller).

Controller (Controlador)

Recebe as requisições HTTP e coordena as ações entre Model e View.

Em Spring Boot, usa anotações como @RestController ou @Controller.

Exemplo Hello World Spring Boot

Fluxo no Spring Boot (API REST típica):

→ Requisição HTTP chega ao Controller

→ Controller chama o Service (lógica de negócio)

→ Service usa o Repository (acesso a banco de dados)

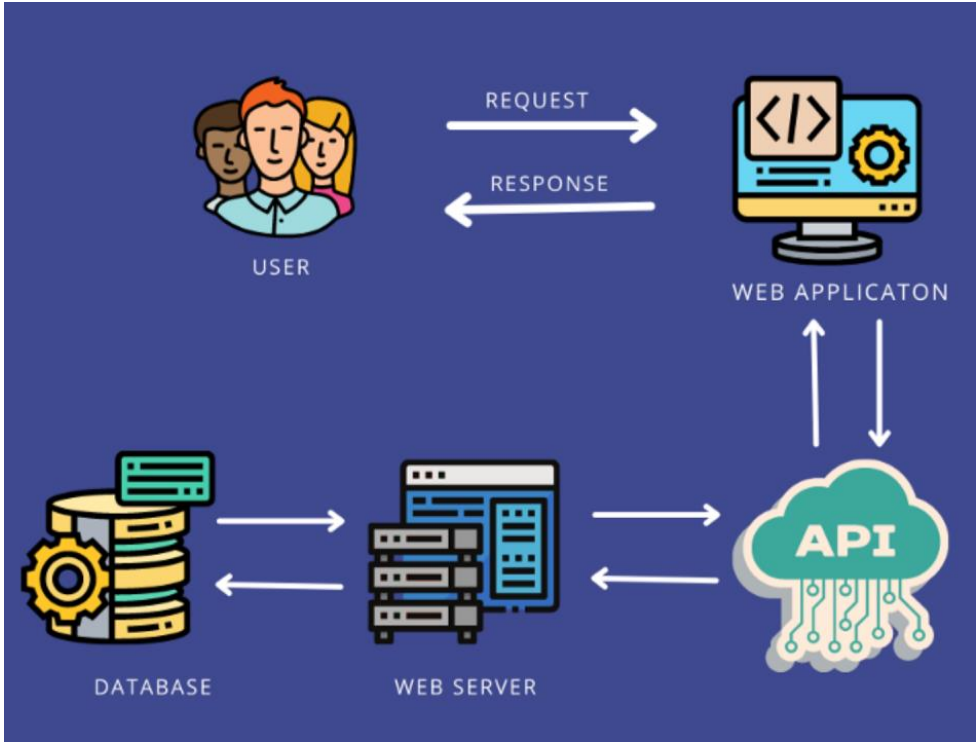
→ Resposta é gerada (JSON/XML) e enviada ao cliente

Exemplo Hello World Spring Boot

MVC vs. REST APIs (Spring Boot moderno):

Em APIs RESTful, a View é substituída por respostas JSON/XML.

O Controller (@RestController) retorna dados diretamente (sem renderizar HTML).



Exemplo Hello World Spring Boot

No package **hello_spring_boot** crie o package **controller** e a classe **HelloController**

```java
package br.com.aweb.hello_spring_boot.controller;

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

@RestController
public class HelloController {
    @GetMapping
    public String sayHello(){
        return "Olá Mundo Spring Boot!";
    }
}
```

Exemplo Hello World Spring Boot

@RestController

- Transforma a classe em um componente gerenciado pelo Spring.
- Os retornos dos métodos são escritos diretamente na resposta HTTP
- Combina @Controller + @ResponseBody em uma única anotação
- @Controller tradicional (que espera retornar um nome de view)
- Todos os métodos retornam diretamente o corpo da resposta (não redireciona para views/templates)

@GetMapping

- Mapeia requisições HTTP GET para este método
- Quando usado sem parâmetro (como no exemplo), responde à raiz do contexto (/)
- Equivalente a @RequestMapping(method = RequestMethod.GET)
- Existem equivalentes para outros verbos HTTP:
  - ✓ @PostMapping
  - ✓ @PutMapping
  - ✓ @DeleteMapping
- O caminho pode ser configurado: @GetMapping("/ola")

Exemplo Hello World Spring Boot

Implemente:

```
13      @GetMapping("/ola")
14      public String sayHelloCustom(){
15          return "Olá endpoint específico!";
16      }
17  }
```

Teste:

https://localhost:8080/ola

Exemplo Hello World Spring Boot

@RequestParam

- É uma anotação do Spring usada para capturar parâmetros passados na URL de uma requisição HTTP (geralmente em requisições GET ou POST com formulários).
- Se o parâmetro não for enviado, o Spring retorna um erro 400 Bad Request.
- Se o parâmetro for opcional, utilizar por exemplo @RequestParam(required = false)
- É possível definir um valor padrão com por exemplo @RequestParam(defaultValue = “Valor padrão”)
- É possível remapear variáveis com por exemplo @RequestParam("username") String nomeUsuario

Exemplo Hello World Spring Boot

Implemente:

```
19      @GetMapping("/greet")
20      public String greet(@RequestParam String name) {
21          return "Olá, " + name + "! Bem-vindo(a)!";
22      }
23  }
```

Teste:

https://localhost:8080/greet?name=André

Exemplo Hello World Spring Boot

Altere:

```
19      @GetMapping("/greet")
20      public String greet(@RequestParam(required = false) String name) {
21          return "Olá, " + name + "! Bem-vindo(a)!";
22      }
23  }
```

Teste:

https://localhost:8080/greet
https://localhost:8080/greet?name=André

Exemplo Hello World Spring Boot

Altere:

```
19      @GetMapping("/greet")
20      public String greet(@RequestParam(defaultValue = "Visitante") String name) {
21          return "Olá, " + name + "! Bem-vindo(a)!";
22      }
23  }
```

Teste:

https://localhost:8080/greet
https://localhost:8080/greet?name=André

Exemplo Hello World Spring Boot

Altere:

```
19      @GetMapping("/greet")
20      public String greet(@RequestParam("name") String userName) {
21          return "Olá, " + userName + "! Bem-vindo(a)!";
22      }
23  }
```

Teste:

https://localhost:8080/greet
https://localhost:8080/greet?name=André

Exercício – Calculadora Simples

Crie um endpoint /calcular que:

- **Aceite três parâmetros:**
  - ✓ num1 (obrigatório, número)
  - ✓ num2 (obrigatório, número)
  - ✓ op (opcional, valores: "soma" ou "subtracao"; padrão = "soma")
- **Retorne o resultado da operação. Exemplos:**
  - ✓ localhost:8080/calcular?num1=5&num2=3 → Retorna: "Resultado: 8" (soma padrão)
  - ✓ localhost:8080/calcular?num1=10&num2=4&op=subtracao → Retorna: "Resultado: 6"

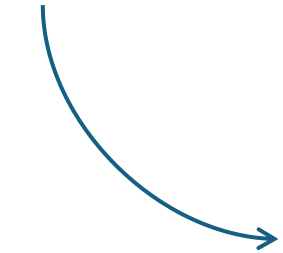
Exercício – Endpoint com Parâmetros Condicionais

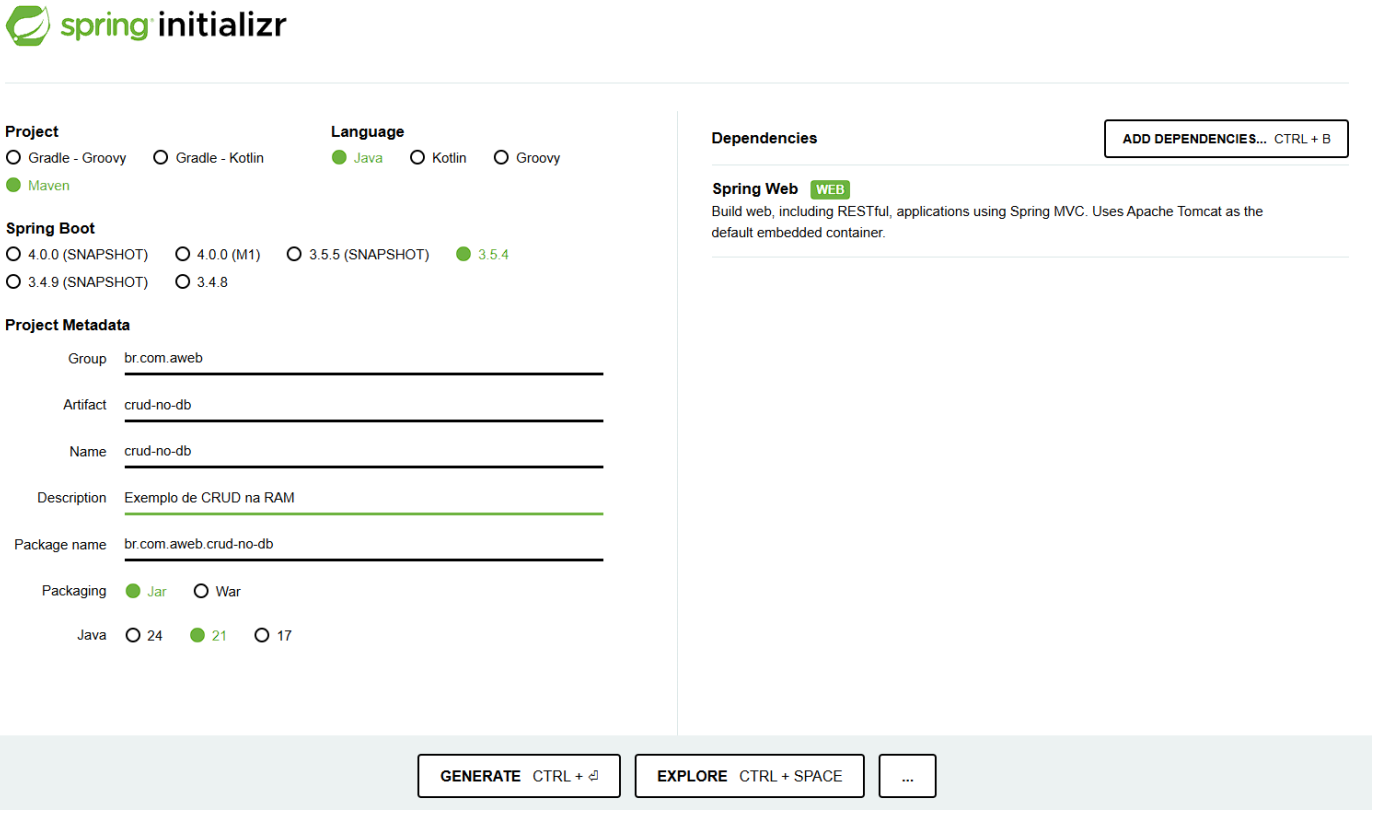
Crie um endpoint /mensagem que:

- **Aceite dois parâmetros opcionais:**
  - ✓ usuario (se não fornecido, use "Visitante")
  - ✓ idioma (valores: "pt" ou "en"; padrão = "pt")
- **Retorne uma mensagem personalizada:**
  - ✓ Se idioma="pt": "Olá, [usuario]! Bem-vindo(a)."
  - ✓ Se idioma="en": "Hello, [usuario]! Welcome."
- **Exemplos:**
  - ✓ localhost:8080/mensagem → Retorna: "Olá, Visitante! Bem-vindo(a)."
  - ✓ localhost:8080/mensagem?usuario=Ana&idioma=en → Retorna: "Hello, Ana! Welcome."

Exemplo CRUD sem DB

start.spring.io





Exemplo CRUD sem DB

Os verbos HTTP (também chamados de métodos HTTP) são a base das APIs RESTful.
Definem a ação que o cliente quer realizar em um recurso específico.
No Spring Boot, esses verbos são mapeados usando anotações.

VERBO	ANOTAÇÃO SPRING	DESCRIÇÃO	CÓDIGO DE STATUS TÍPICO
GET	@GetMapping	Solicita dados de um recurso.	200 (OK)
POST	@PostMapping	Cria um novo recurso ou processa dados.	201 (Created)
PUT	@PutMapping	**Substitui** um recurso existente ou cria se não existir.	200 (OK) ou 204 (No Content)
PATCH	@PatchMapping	Atualiza **parcialmente** um recurso.	200 (OK)
DELETE	@DeleteMapping	Remove um recurso.	204 (No Content)

Exemplo CRUD sem DB

@RequestBody

Anotação do Spring que **desserializa** o corpo de uma requisição HTTP (JSON/XML) em um objeto Java.

Comparação com @RequestParam

Feature	@RequestParam	@RequestBody
Origem	Parâmetros da URL (?chave=valor)	Corpo da requisição (JSON/XML)
Uso típico	GET	POST/PUT/PATCH
Complexidade	Dados simples	Dados estruturados (objetos)

Exemplo de JSON →

```
{
  "nome": "João",
  "email": "joao@email.com"
}
```

Exemplo CRUD sem DB

O que é um DTO (Data Transfer Object)?

Padrão de design que define um objeto exclusivamente para transferência de dados entre camadas (ex: Controller ↔ Frontend).

Por que usar?

Evita expor a entidade de banco de dados diretamente na API.

Permite customizar os dados trafegados (ex: máscaras, campos calculados).

@PathVariable

É uma anotação do Spring usada para extrair valores de trechos dinâmicos da URL e passá-los como parâmetros para os métodos do controlador, útil para operações que envolvem IDs ou identificadores únicos.

Utilizado em operações CRUD onde você precisa identificar um recurso específico:

GET /produtos/{id} → Buscar produto por ID.
PUT /produtos/{id} → Atualizar produto por ID.
DELETE /produtos/{id} → Remover produto por ID.

Exemplo CRUD sem DB

No package **crud_no_db** crie o package **dto** e a classe **ProductDTO**.

```java
package br.com.aweb.crud_no_db.dto;

public class ProductDTO {
    private Long id;
    private String name;
    private Double price;

    // construtor vazio
    public ProductDTO() { }

    // métodos getters e setters para todos atributos
```

Exemplo CRUD sem DB

No package **crud_no_db** crie o package **controller** e a classe **ProductController**.

```java
package br.com.aweb.crud_no_db.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

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.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.aweb.crud_no_db.dto.ProductDTO;
```

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

    private Map<Long, ProductDTO> products = new HashMap<>();
    private Long nextId = 1L;

    // listar todos produtos
    @GetMapping
    public List<ProductDTO> allProducts() {
        return new ArrayList<>(products.values());
    }

    // buscar produto por id
    @GetMapping("/{id}")
    public ProductDTO getProductById(@PathVariable Long id) {
        return products.get(id);
    }
```

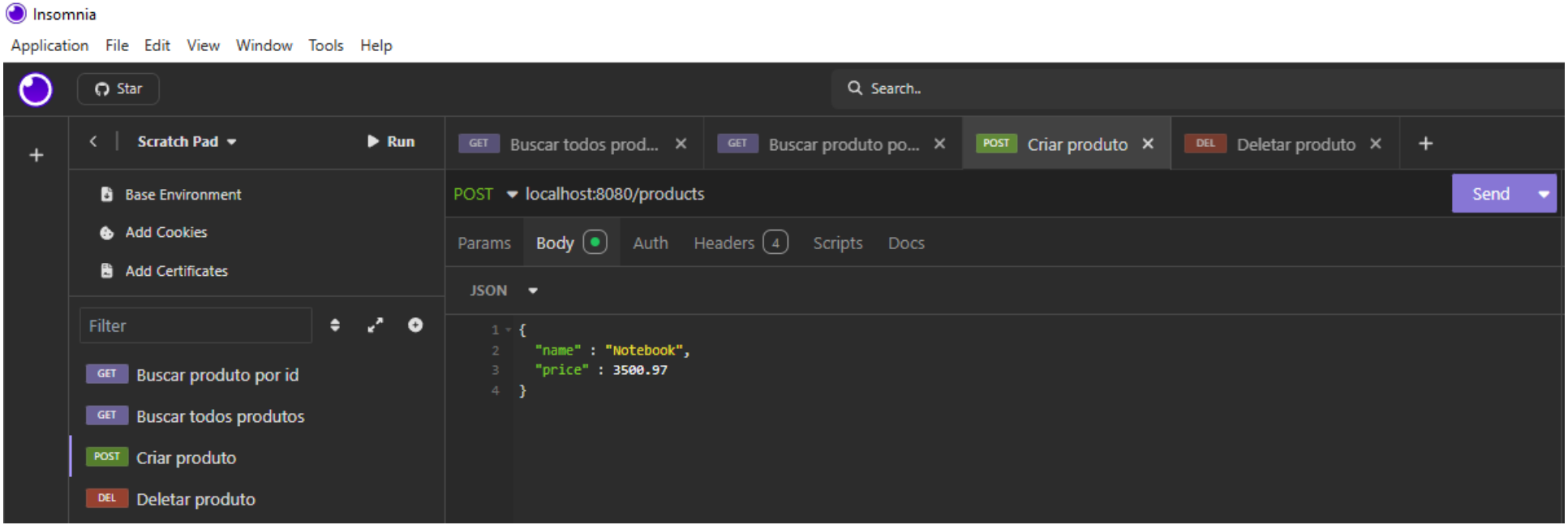
Exemplo CRUD sem DB

```java
    // criar produto
    @PostMapping
    public ProductDTO createProduct(@RequestBody ProductDTO product) {
        product.setId(nextId++);
        products.put(product.getId(), product);
        return product;
    }

    // remover produto
    @DeleteMapping("/{id}")
    public String deleteProduct(@PathVariable Long id) {
        if (products.remove(id) != null)
            return "Produto removido!";
        return "Produto não encontrado!";
    }
}
```

Exemplo CRUD sem DB

Teste a API desenvolvida



Exercício – CRUD sem DB

Inclua o método PUT, através da anotação @PutMapping, desenvolva um endpoint para atualização de um produto.

Exercícios – Regras Comuns a Todos Exercícios

Não use banco de dados (armazene em Map ou List).

Mantenha os projetos isolados (um novo projeto Spring Boot para cada exercício).

Exercício – Sistema de Usuários Básico

Crie um endpoint /usuarios que:

- POST Receba JSON com { "nome": String, "email": String }
  - ✓ Atribua um ID automático

- GET Liste todos os usuários cadastrados

Exercício – Catálogo de Livros

Crie um endpoint /livros que:

- POST Adicione livros { "titulo": String, "autor": String }
  ✓ Atribua um ID automático
- PUT Atualize TODOS os campos por ID (substitua o livro inteiro)
- GET /livros/{id}: Busque por ID

Exercício – API de Comentários

Crie um endpoint /comentarios que:

- POST Adicione { "autor": String, "texto": String }
  - ✓ Atribua um ID automático
- GET Liste todos comentários
- GET/{id} Busque por ID
- PUT/{id} Atualize texto (mantendo o autor original)
- DELETE /{id} Remova o comentário

Exercício – Sistema de Reservas

Crie um endpoint /reservas que:

- POST Crie reservas { "cliente": String, "data": String (dd/mm/aaaa) }
  ✓ Atribua um ID automático
- GET /cliente/{nome} Liste todas as reservas de um cliente
- DELETE /{id} Cancele uma reserva

Exercício – Gerenciador de Tarefas Avançado

Crie um endpoint /tarefas que:

- POST Adicione { "descricao": String, "prioridade": "alta/media/baixa" }
  - ✓ Atribua um ID automático
- GET Filtre por prioridade (ex: /tarefas?prioridade=alta)
- PUT /{id}/status Atualize apenas o status (ex: { "status": "concluída" })
- DELETE /{id} Remova a tarefa