Etec
Bento Quirino
Campinas

Tutorial: Construindo APIs REST em Java com Spring Boot

Justificativa

Desenvolver APIs REST em Java proporciona uma base sólida para aplicações web modernas. Com essa habilidade, você entenderá conceitos fundamentais de backend, banco de dados e integração entre sistemas, além de aprender uma das linguagens mais utilizadas no mercado: **Java**.

Objetivos

- Compreender o que é uma API REST e sua função.
- Aprender a criar uma API em Java usando o **Spring Boot**.
- Realizar operações de **CRUD (Create, Read, Update, Delete)**.
- Conectar a API ao banco de dados **MySQL**.
- Desenvolver e testar o projeto no **VSCode**.
- Utilizar os principais métodos do **Spring Data JPA**.

Índice

1. O que é uma API REST?
2. Tecnologias utilizadas
3. Preparando o ambiente
4. Criando o projeto Spring Boot
    - Opção 1: Spring Initializr (recomendada)
    - Opção 2: Linha de comando
5. Estrutura do projeto
6. Criando a entidade Produto
7. Criando o repositório com Spring Data JPA
8. Criando o controller REST
9. Configurando o banco de dados MySQL
10. Testando os endpoints com Postman
11. Compilando e executando no VSCode
12. Principais métodos JPA
13. Conclusão
14. Referências

---

1. O que é uma API REST?

API (Application Programming Interface) é um conjunto de regras que permite que programas diferentes se comuniquem. Uma API **REST** segue princípios como uso do protocolo **HTTP**, uso de **JSON** para dados e divisão em **endpoints** (URLs que representam ações).

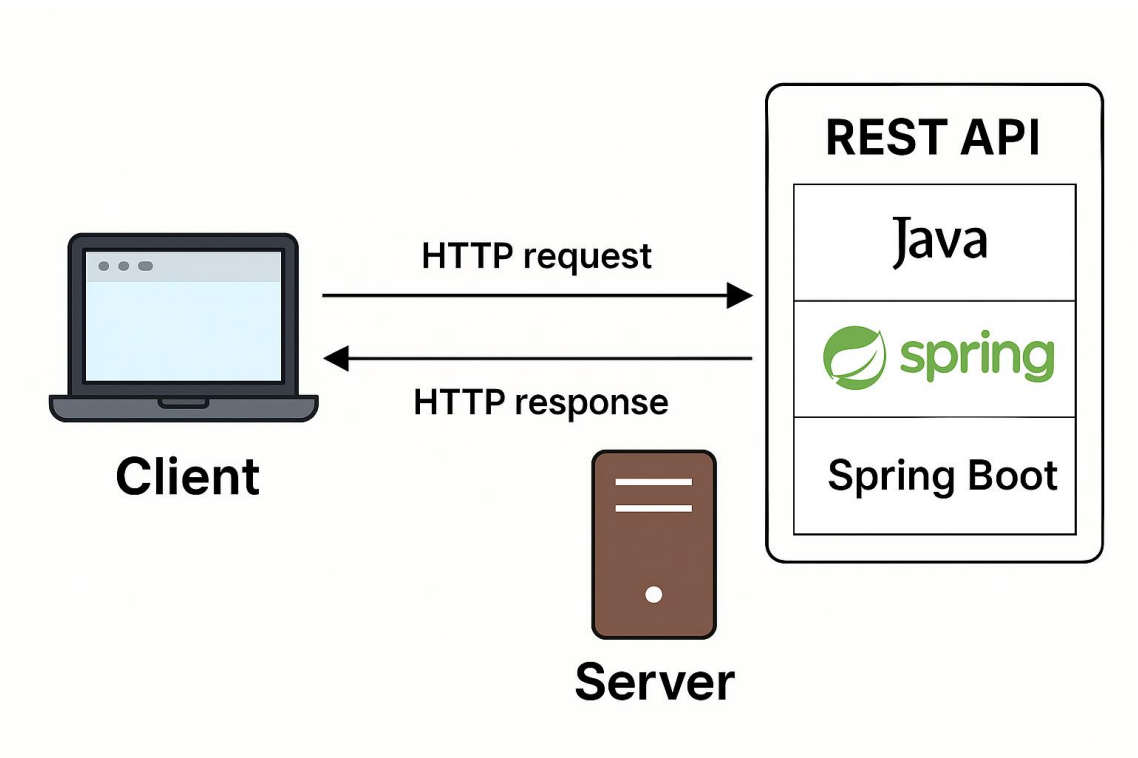

Figura 1 - elaboração própria

---

2. Tecnologias Utilizadas

- **Java 17 ou superior**
- **Spring Boot** – Framework para aplicações Java modernas
- **Spring Data JPA** – Facilita o acesso ao banco de dados
- **MySQL** – Banco de dados relacional
- **Maven** – Gerenciador de dependências
- **VSCode** – Editor de código

Extensões para o VSCode:

- **Extension Pack for Java**
- **Spring Boot Extension Pack**
- **Debugger for Java**
- **Maven for Java**

---

3. Preparando o Ambiente

1. Instale o **Java 17 JDK**: https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html
2. Instale o **MySQL**: https://dev.mysql.com/downloads/installer/
3. Instale o **VSCode**: https://code.visualstudio.com/
4. Instale as extensões recomendadas acima.
5. Instale o **Maven** (caso necessário): https://maven.apache.org/download.cgi

---

4. Criando o Projeto Spring Boot

Opção 1 – Spring Initializr (Recomendada)

1. Acesse: https://start.spring.io
2. Configure:
   - Project: Maven
   - Language: Java
   - Spring Boot: 3.x
   - Group: com.exemplo
   - Artifact: api-produtos
3. Adicione as dependências:
   - Spring Web
   - Spring Data JPA
   - MySQL Driver
4. Clique em **"Generate"** e extraia o ZIP.

Abra a pasta no VSCode.

Opção 2 – Linha de Comando

Se você já tem Maven instalado, crie o projeto com:

```
mvn archetype:generate -DgroupId=com.exemplo -DartifactId=api-produtos \
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Depois adicione manualmente as dependências no pom.xml.

---

5. Estrutura do Projeto

```
src/
└── main/
    ├── java/com/exemplo/
    │   ├── ApiProdutosApplication.java
    │   ├── model/Produto.java
    │   ├── repository/ProdutoRepository.java
    │   └── controller/ProdutoController.java
    └── resources/
        └── application.properties
```

---

6. Criando a Entidade Produto

Arquivo: model/Produto.java

```java
package com.bentao.api_produtos.model;

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

@Entity
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private Double preco;

    public Produto() {}

    public Produto(String nome, Double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    // Getters e Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
    public Double getPreco() { return preco; }
    public void setPreco(Double preco) { this.preco = preco; }

}

```

7. Criando o Repositório

Arquivo: repository/ProdutoRepository.java

```java
package com.bentao.api_produtos.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import com.bentao.api_produtos.model.Produto;

public interface ProdutoRepository extends JpaRepository<Produto, Long> {
    List<Produto> findByNome(String nome);
    List<Produto> findByNomeContaining(String parteDoNome);
}
```

8. Criando o Controller

Arquivo: controller/ProdutoController.java

```java
package com.bentao.api_produtos.controller;

import java.util.List;

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 com.bentao.api_produtos.model.Produto;
import com.bentao.api_produtos.repository.ProdutoRepository;

@RestController
@RequestMapping("/produtos")
public class ProdutoController {

    private final ProdutoRepository repository;

    public ProdutoController(ProdutoRepository repository) {
        this.repository = repository;
    }
```

```java
@PostMapping
public Produto criar(@RequestBody Produto produto) {
    return repository.save(produto);
}

@GetMapping
public List<Produto> listar() {
    return repository.findAll();
}

@GetMapping("/{id}")
public Produto buscar(@PathVariable Long id) {
    return repository.findById(id).orElse(null);
}
```

```java
    @PutMapping("/{id}")
    public Produto atualizar(@PathVariable Long id,
                             @RequestBody Produto novoProduto) {
        return repository.findById(id).map(produto -> {
            produto.setNome(novoProduto.getNome());
            produto.setPreco(novoProduto.getPreco());
            return repository.save(produto);
        }).orElse(null);
    }

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

9. Configurando o MySQL

Arquivo: src/main/resources/application.properties

```
spring.application.name=api-produtos
spring.datasource.url=jdbc:mysql://localhost:3306/seubanco
spring.datasource.username=seuusuario
spring.datasource.password=suasenha
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Substitua seubanco, seuusuario e suasenha pelas suas informações.

---

10. Testando com o Postman

Exemplos:

- **POST** http://localhost:8080/produtos

```
{
  "nome": "Notebook",
  "preco": 3200.00
}
```

- **GET** http://localhost:8080/produtos
- **GET** http://localhost:8080/produtos/1
- **PUT** http://localhost:8080/produtos/1
- **DELETE** http://localhost:8080/produtos/1

---

11. Compilando e Executando no VSCode

Passos:

1. **Abrir a pasta do projeto no VSCode.**
2. **Compilar o projeto:**

```
./mvnw clean install
```

ou no Windows:

```
mvnw.cmd clean install
```

3. **Executar:**

```
./mvnw spring-boot:run
```

A aplicação será acessível em:
http://localhost:8080

---

12. Principais Métodos do Spring Data JPA

1. CRUD – Métodos básicos para criar, ler, atualizar e deletar

Método	Descrição
save(entity)	Salva ou atualiza um objeto no banco de dados.
saveAll(Iterable)	Salva uma lista de objetos.
findById(id)	Busca um objeto pelo ID. Retorna Optional<T>.
findAll()	Retorna todos os registros da tabela.
findAllById(Iterable<Id>)	Retorna todos os registros com os IDs especificados.
deleteById(id)	Deleta o objeto com o ID fornecido.
delete(entity)	Deleta o objeto fornecido.
deleteAll()	Deleta todos os registros.
count()	Retorna a quantidade total de registros na tabela.
existsById(id)	Verifica se existe um registro com o ID fornecido. Retorna true/false.

---

2. Consultas com palavras-chave (Query Methods)

Esses métodos são criados com base no nome — o Spring entende o que você quer fazer e monta a SQL automaticamente:

Nome do Método	O que faz
findByNome(String nome)	Busca por nome exato

Nome do Método	O que faz
findByNomeContaining(String nome)	Busca por nome que **contém** (equivalente ao SQL LIKE %x%)
findByPrecoGreaterThan(Double x)	Preço maior que
findByPrecoLessThan(Double x)	Preço menor que
findByAtivoTrue()	Busca onde o campo booleano ativo for true
findTop3ByOrderByPrecoDesc()	Retorna os 3 mais caros
findByCategoriaAndPreco()	Filtra por dois campos ao mesmo tempo

---

3. Paginação e Ordenação

Esses métodos permitem paginar resultados e ordenar sem precisar escrever SQL.

```
Page<Produto> findAll(Pageable pageable);  // retorna dados paginados
List<Produto> findAll(Sort sort);          // retorna todos ordenados
```

Exemplo de uso:

```
PageRequest.of(0, 5, Sort.by("preco").descending());
```

- Página 0
- 5 itens por página
- Ordenado por preço (decrescente)

---

4. Consultas com @Query personalizada (quando o nome do método não basta)

Quando os métodos baseados em nomes não forem suficientes, você pode escrever sua própria consulta:

```
@Query("SELECT p FROM Produto p WHERE p.nome LIKE %:nome%")
List<Produto> buscarPorNome(@Param("nome") String nome);
```

Ou, se quiser usar SQL nativo:

```
@Query(value = "SELECT * FROM produtos WHERE preco > :preco", nativeQuery = true)
List<Produto> buscarProdutosCaros(@Param("preco") Double preco);
```

Dica: Combine palavras para criar métodos poderosos

Exemplo	Resultado
findByNomeAndPrecoGreaterThan(String, Double)	Produtos com nome X e preço acima de Y
findByNomeOrCategoria(String, String)	Produtos com nome X **ou** categoria Y
findByNomeIgnoreCase(String nome)	Busca por nome ignorando letras maiúsculas/minúsculas

---

13. Conclusão

Você aprendeu a criar, configurar e testar uma API REST com Java e Spring Boot no VSCode, usando o MySQL como banco de dados. Também conheceu métodos importantes do Spring Data JPA e viu como implementar suas próprias consultas personalizadas.

Esse conhecimento é a base para desenvolver sistemas completos com frontend e backend separados, APIs seguras e integrações com outras plataformas. Continue explorando!

---

14. Referências

SPRING. *Spring Boot*. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 12 maio 2025.

ALURA. *O que é Spring Boot e como dar os seus primeiros passos*. Disponível em: https://www.alura.com.br/artigos/spring-boot?srsltid=AfmBOoqSTj4O95sY8MnwKU8tFRvY7Ze_YihFS-OV214Mif18MfzsdpDV. Acesso em: 12 maio 2025.

ALURA. *Java: persistência de dados e consultas com Spring Data JPA*. Disponível em: https://www.alura.com.br/conteudo/java-persistencia-dados-consultas-spring-data-jpa?srsltid=AfmBOoodfwRZT1aGfTSc1FQXn_IzrCdfFHhuxXMr1aA0AZw8iWpF4pwi. Acesso em: 12 maio 2025.

MARX, M. *Melhores práticas básicas da API de persistência Java*. Oracle. Disponível em: https://www.oracle.com/technical-resources/articles/javaee/marx-jpa.html. Acesso em: 12 maio 2025.

MYSQL. *MySQL – Documentação*. Disponível em: https://dev.mysql.com/doc/refman/8.4/en/. Acesso em: 12 maio 2025.

POSTMAN. *Postman – Documentação*. Disponível em: https://learning.postman.com/docs/introduction/overview/. Acesso em: 12 maio 2025.