API para gestão de livros

(incluindo **comentários explicativos**, **notas pedagógicas** e **testes**)

## Passo-a-Passo para Implementar a API

### 1. Criar o Projeto com Spring Initializr

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

**Nota:** não esquecer as dependências (lado direito).

A estrutura do projeto, com as sub-pastas da pasta “src” todas abertas deve ser como indicado na imagem em baixo:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

### 2. Configurar o Application.properties

1. Configure o ficheiro application.properties para o 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

1. Se usar outra base de dados, altere a url, username, password, e a database-platform.

### 3. Criar a Entidade Livro

* **Explicação**: A classe Livro será mapeada para uma tabela na base de dados. Use a anotação @Entity para indicar que esta classe representa uma entidade JPA.

Código com comentários:

package com.example.livros.model;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class Livro {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY) // Geração automática do ID.  
 private Long id;  
 private String titulo; // Título do livro.  
 private String autor; // Autor do livro.  
  
 // Construtor vazio é necessário para o JPA.  
 public Livro() {}  
  
 // Construtor para facilitar a criação de instâncias.  
 public Livro(String titulo, String autor) {  
 this.titulo = titulo;  
 this.autor = autor;  
 }  
  
 // Getters e Setters  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getTitulo() {  
 return titulo;  
 }  
  
 public void setTitulo(String titulo) {  
 this.titulo = titulo;  
 }  
  
 public String getAutor() {  
 return autor;  
 }  
  
 public void setAutor(String autor) {  
 this.autor = autor;  
 }  
}

### 4. Criar o Repositório LivroRepository

* **Explicação**: A interface LivroRepository permite o acesso direto à base de dados, herdando métodos úteis como save(), findAll(), findById(), entre outros.

Código:

package com.example.livros.repository;  
  
import com.example.livros.model.Livro;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface LivroRepository extends JpaRepository<Livro, Long> {  
 // Métodos adicionais podem ser definidos aqui, se necessário.  
}

### 5. Criar o Serviço LivroService

* **Explicação**: A camada de serviço centraliza a lógica de negócio. Este é o local ideal para incluir validações e regras adicionais.

Código com comentários:

package com.example.livros.service;  
  
import com.example.livros.model.Livro;  
import com.example.livros.repository.LivroRepository;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class LivroService {  
  
 private final LivroRepository livroRepository;  
  
 // Injeta o repositório.  
 public LivroService(LivroRepository livroRepository) {  
 this.livroRepository = livroRepository;  
 }  
  
 // Listar todos os livros.  
 public List<Livro> listarTodos() {  
 return livroRepository.findAll();  
 }  
  
 // Procurar um livro pelo ID.  
 public Livro procurarPorId(Long id) {  
 Optional<Livro> livro = livroRepository.findById(id);  
 return livro.orElseThrow(() -> new RuntimeException("Livro não encontrado"));  
 }  
  
 // Adicionar ou atualizar um livro.  
 public Livro salvarOuAtualizar(Livro livro) {  
 return livroRepository.save(livro);  
 }  
  
 // Remover um livro pelo ID.  
 public void remover(Long id) {  
 if (!livroRepository.existsById(id)) {  
 throw new RuntimeException("Livro não encontrado");  
 }  
 livroRepository.deleteById(id);  
 }  
}

### 6. Criar o Controlador LivroController

* **Explicação**: O controlador é responsável por expor a API REST.

Código com comentários:

package com.example.livros.controller;  
  
import com.example.livros.model.Livro;  
import com.example.livros.service.LivroService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/livros")  
public class LivroController {  
  
 private final LivroService livroService;  
  
 // Injeta o serviço.  
 public LivroController(LivroService livroService) {  
 this.livroService = livroService;  
 }  
  
 // Listar todos os livros.  
 @GetMapping  
 public List<Livro> listarTodos() {  
 return livroService.listarTodos();  
 }  
  
 // Procurar um livro pelo ID.  
 @GetMapping("/{id}")  
 public ResponseEntity<Livro> procurarPorId(@PathVariable Long id) {  
 try {  
 Livro livro = livroService.procurarPorId(id);  
 return ResponseEntity.ok(livro);  
 } catch (RuntimeException e) {  
 return ResponseEntity.notFound().build();  
 }  
 }  
  
 // Adicionar ou atualizar um livro.  
 @PostMapping  
 public Livro salvar(@RequestBody Livro livro) {  
 return livroService.salvarOuAtualizar(livro);  
 }  
  
 // Remover um livro pelo ID.  
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void> remover(@PathVariable Long id) {  
 try {  
 livroService.remover(id);  
 return ResponseEntity.noContent().build();  
 } catch (RuntimeException e) {  
 return ResponseEntity.notFound().build();  
 }  
 }  
}

### 7. Criar um DataLoader

* **Explicação**: O objetivo é colocarmos alguns livros na base de dados, no arranque do programa.

Código com comentários:

package com.example.livros;

import com.example.livros.model.Livro;

import com.example.livros.repository.LivroRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class DataLoader {

    @Bean

    public CommandLineRunner carregarDados(LivroRepository livroRepository) {

        return args -> {

            // Lista de livros iniciais

            livroRepository.save(new Livro("1984", "George Orwell"));

            livroRepository.save(new Livro("A Revolução dos Bichos", "George Orwell"));

            livroRepository.save(new Livro("Dom Quixote", "Miguel de Cervantes"));

            livroRepository.save(new Livro("Moby Dick", "Herman Melville"));

            livroRepository.save(new Livro("Guerra e Paz", "Lev Tolstói"));

            livroRepository.save(new Livro("Orgulho e Preconceito", "Jane Austen"));

            livroRepository.save(new Livro("Crime e Castigo", "Fiódor Dostoiévski"));

            livroRepository.save(new Livro("O Senhor dos Anéis", "J.R.R. Tolkien"));

            livroRepository.save(new Livro("O Hobbit", "J.R.R. Tolkien"));

            livroRepository.save(new Livro("Cem Anos de Solidão", "Gabriel García Márquez"));

            livroRepository.save(new Livro("O Grande Gatsby", "F. Scott Fitzgerald"));

            livroRepository.save(new Livro("Ulisses", "James Joyce"));

            livroRepository.save(new Livro("A Metamorfose", "Franz Kafka"));

            livroRepository.save(new Livro("O Estrangeiro", "Albert Camus"));

            livroRepository.save(new Livro("A Divina Comédia", "Dante Alighieri"));

            livroRepository.save(new Livro("Hamlet", "William Shakespeare"));

            livroRepository.save(new Livro("O Retrato de Dorian Gray", "Oscar Wilde"));

            livroRepository.save(new Livro("Jane Eyre", "Charlotte Brontë"));

            livroRepository.save(new Livro("As Aventuras de Huckleberry Finn", "Mark Twain"));

            livroRepository.save(new Livro("Drácula", "Bram Stoker"));

            System.out.println("Dados iniciais carregados com sucesso!");

        };

    }

}

### 7. Testar os Endpoints

1. Abra o terminal ou uma ferramenta como o **Postman**:
2. Teste os seguintes endpoints:
   * **Listar todos os livros**: GET http://localhost:8080/api/livros
   * **Procurar um livro pelo ID**: GET http://localhost:8080/api/livros/{id}
   * **Adicionar um livro**: POST http://localhost:8080/api/livros com corpo JSON:
   * {  
      "titulo": "1984",  
      "autor": "George Orwell"  
     }
   * **Remover um livro**: DELETE http://localhost:8080/api/livros/{id}.

### 

### Notas sobre (não uso de) @JsonManagedReference e @JsonBackReference

Ao contrário de outros exemplos do curso, aqui não usamos estas anotações: elas são necessárias apenas para evitar loops infinitos em relações bidirecionais, como @OneToMany e @ManyToOne. Como este exemplo não possui relações desse tipo, elas não são necessárias. Se os alunos adicionarem relações mais complexas no futuro, devem considerar estas anotações.