POÇÕES REST

A Arte de Criar APIs Encantadas

Misture código, lógica e magia para dar vida aos seus primeiros endpoints

Whesley Kallil

Introdução

Criando um CRUD simples

Neste eBook, vamos construir um CRUD mágico usando **Java 21**, **Spring Boot 3**, e **PostgreSQL**. Nosso objetivo é criar uma API simples para gerenciar **alunos da Escola de Magia de Hogwarts**.

Cada aluno terá:

- **Name**
- **age**
- **characteristics (com afinidades de 0 a 10)**
- **house (definida automaticamente de acordo com as características)**

Você pode gerar o projeto em **https://start.spring.io** com as dependências:

- **Spring Web**
- **Spring Data JPA**
- **PostgreSQL Driver**

01

CONFIGURANDO O BANDO DE DADOS

DATABASE SETUP

Altere o **application.properties** para **application.yml**, e escreva as configurações abaixo, alterando os dados como você deseja:

Applitcation.yml

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/hogwarts
    username: postgres
    password: 1234
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

02

CRIANDO A ENTIDADE STUDENT

CAMADA ENTITY

Antes de qualquer magia acontecer, precisamos definir **quem é nosso aluno** dentro do sistema.

A classe ***Student*** representa o aluno de Hogwarts com seus atributos principais: nome, idade, características e a casa que o chapéu seletor vai escolher.

Essa entidade será a base do nosso CRUD.

Applitcation.yml

```java
@Entity
public class Student {

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

    private String name;
    private int age;

    private int intelligence;
    private int courage;
    private int loyalty;
    private int ambition;

    private String house;

    // Getters e setters
}
```

03

DEFININDO O REPOSITÓRIO

CAMADA REPOSITORY

Com o aluno criado, precisamos de um jeito de **guardar e buscar** essas informações no banco.

O repositório é responsável por conversar com o PostgreSQL, permitindo salvar, listar, atualizar e deletar alunos de forma simples usando o poder do **Spring Data JPA**.

Crie uma **interface** ao invés de uma class.

StudentRepository

```java
public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

04

LÓGICA DO CHAPÉU SELETOR

CAMADA SERVICE

Agora vem a parte mais divertida!
Vamos ensinar o sistema a agir como o **Chapéu Seletor**, atribuindo cada aluno à casa que mais combina com suas características (coragem, inteligência, lealdade ou ambição).
Aqui, a lógica mágica acontece antes de salvar o aluno no banco.

StudentService

```java
@Service
public class StudentService {

    private final StudentRepository repository;

    public StudentService(StudentRepository repository) {
        this.repository = repository;
    }

    public Student createStudent(Student student) {
        student.setHouse(assignHouse(student));
        return repository.save(student);
    }
```

CONTINUAÇÃO SERVICE

StudentService

```java
    private String assignHouse(Student s) {
        int gryffindor = s.getCourage();
        int ravenclaw = s.getIntelligence();
        int hufflepuff = s.getLoyalty();
        int slytherin = s.getAmbition();

        int max = Math.max(
            Math.max(gryffindor, ravenclaw),
            Math.max(hufflepuff, slytherin)
            );

        if (max == gryffindor) return "Gryffindor";
        if (max == ravenclaw) return "Ravenclaw";
        if (max == hufflepuff) return "Hufflepuff";
        return "Slytherin";
    }

    public List<Student> listStudents() {
        return repository.findAll();
    }

    public Student updateStudent(Long id, String name, Integer age) {
        Student s = repository.findById(id)
                .orElseThrow(() -> new RuntimeException("Student not found"));

        if (name != null) s.setName(name);
        if (age != null) s.setAge(age);

        return repository.save(s);
    }

    public void deleteStudent(Long id) {
        repository.deleteById(id);
    }
}
```

05

CRIANDO O CONTROLLER

CAMADA CONTROLLER

Com a lógica pronta, é hora de abrir as portas do castelo!

O ***Controller*** será a entrada da nossa API, é ele quem recebe os feitiços (**requisições HTTP**) e os direciona para os serviços corretos.

A partir daqui, qualquer bruxo ou trouxa poderá interagir com o CRUD de Hogwarts.

StudentController

```java
@RestController
@RequestMapping("/students")
public class StudentController {

    private final StudentService service;

    public StudentController(StudentService service) {
        this.service = service;
    }

    @PostMapping
    public Student create(@RequestBody Student student) {
        return service.createStudent(student);
    }
```

CONTINUAÇÃO CONTROLLER

StudentController

```java
    @GetMapping
    public List<Student> list() {
        return service.listStudents();
    }

    @PutMapping("/{id}")
    public Student update(@PathVariable Long id,
                          @RequestParam(required = false) String name,
                          @RequestParam(required = false) Integer age) {
        return service.updateStudent(id, name, age);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        service.deleteStudent(id);
    }
}
```

06

TESTANDO OS ENDPOINTS

TESTANDO A API

Use programas como o **Postman** ou **Insomnia** para testar os endpoints criados na camada controller.

Fique com um exemplo de teste do endpoint **POST** ***/students*** que cria um aluno e o retorno esperado:

Postman

```
{
    "name": "Harry Potter",
    "age": 11,
    "intelligence": 7,
    "courage": 9,
    "loyalty": 6,
    "ambition": 5
}
```

RETORNO ESPERADO

Postman

```
{
    "id": 1,
    "name": "Harry Potter",
    "age": 11,
    "house": "Gryffindor"
}
```

07

CONCLUSÃO

PARABÉNS!

Você acabou de criar um **CRUD completo em Java 21 + Spring Boot + PostgreSQL**, com uma pitada mágica de Hogwarts.

Agora você sabe como:

- **Criar uma API simples**
- **Conectar ao banco de dados**
- **Implementar regras de negócio (como o “chapéu seletor”)**

Daqui pra frente, é só expandir: adicione professores, matérias, e até feitiços!

AGRADECIMENTOS

OBRIGADO POR LER ATÉ AQUI

Este eBook foi criado com o propósito de **ensinar de forma leve e prática** conceitos fundamentais de desenvolvimento backend com Java e Spring Boot.

Agradecemos por acompanhar essa jornada mágica de código e aprendizado!

“Esse Ebook foi gerado com ajuda da IA, e diagramado por humano.” 🧙‍♂️

Material produzido apenas para fins didáticos.

https://github.com/wkallil/ebook-crud-simples-hp