Documentação: Criando um CRUD completo com Spring Boot e JPA

## 📌 Objetivo

Desenvolver uma API REST simples com Spring Boot para gerenciar funcionários (Employee), utilizando as camadas:  
- Entidade (Model)  
- Repositório (Repository)  
- Serviço (Service)  
- Controlador REST (Controller)  
- Testes via Postman

## 🧱 1. Criação da Entidade Employee

A camada de modelo (ou entidade) representa a estrutura da tabela no banco de dados. Vamos usar JPA para fazer esse mapeamento automaticamente.  
  
@Entity  
@Table(name = "employee")  
public class Employee { ... }

💡 Use @Entity para indicar que essa classe será persistida no banco e @Table para definir o nome da tabela.

## 🧭 2. Criação do Repositório (interface)

O repositório herda de *JpaRepository*, que oferece todos os métodos prontos de CRUD.  
  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

## ⚙️ 3. Criação da Camada de Serviço

A camada de serviço centraliza as regras de negócio, validações e intermediários entre controller e banco.  
  
@Service  
public class EmployeeService { ... }

## 🌐 4. Criação do Controlador REST

O controlador expõe os endpoints REST para consumo externo, como via Postman, frontend, aplicativos etc.  
  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController { ... }

💡 Os métodos @GetMapping, @PostMapping, etc., são atalhos para os métodos HTTP.

## 📡 5. Testando a API no Postman

🔶 Headers:

* Content-Type: application/json

### 🔁 Endpoints e exemplos:

| **Ação** | **Método** | **URL** | **Corpo JSON** |
| --- | --- | --- | --- |
| Listar todos | GET | /api/employees | – |
| Buscar por ID | GET | /api/employees/1 | – |
| Criar funcionário | POST | /api/employees | Ver abaixo |
| Atualizar | PUT | /api/employees/1 | Ver abaixo |
| Excluir | DELETE | /api/employees/1 | – |

Exemplo de JSON para POST:  
{  
 "name": "João da Silva",  
 "cpf": "123.456.789-00",  
 "email": "joao@empresa.com",  
 "phone": "(11) 91234-5678",  
 "username": "joaos",  
 "password": "senha123",  
 "isActive": true  
}

## 📘 Conclusão

Esse padrão separa responsabilidades e facilita a manutenção. A partir dessa base é possível incluir:  
- Validações (@Valid)  
- DTOs  
- Segurança com Spring Security  
- Paginação e filtros