Para fazer um CRUD (Create, Read, Update, Delete) utilizando Spring Boot, IntelliJ IDEA e PostgreSQL, siga os passos abaixo:

### Passo 1: Criar um Novo Projeto Spring Boot

1. \*\*Abra o IntelliJ IDEA\*\* e crie um novo projeto Spring Boot:

- Vá em \*\*File > New > Project\*\*.

- Escolha \*\*Spring Initializr\*\*.

- Defina o \*\*Group\*\* e \*\*Artifact\*\* (exemplo: `com.example` e `crudapp`).

- Selecione a versão do Java que você está usando.

- Selecione as dependências:

- \*\*Spring Web\*\*

- \*\*Spring Data JPA\*\*

- \*\*PostgreSQL Driver\*\*

- (Opcional) \*\*Spring Boot DevTools\*\*

2. \*\*Gere o projeto\*\* e importe-o no IntelliJ IDEA.

### Passo 2: Configurar o PostgreSQL

1. \*\*Instale o PostgreSQL\*\* se ainda não tiver instalado.

2. \*\*Crie um banco de dados\*\*:

```sql

CREATE DATABASE crud\_app;

```

### Passo 3: Configurar o `application.properties` ou `application.yml`

Configure as informações de conexão com o banco de dados PostgreSQL no arquivo `src/main/resources/application.properties`:

```properties

spring.datasource.url=jdbc:postgresql://localhost:5432/crud\_app

spring.datasource.username=seu\_usuario

spring.datasource.password=sua\_senha

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

```

### Passo 4: Criar o Modelo

Crie uma entidade para representar uma tabela no banco de dados. Exemplo: `src/main/java/com/example/crudapp/model/Produto.java`.

```java

package com.example.crudapp.model;

import jakarta.persistence.\*;

@Entity

public class Produto {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String nome;

private double preco;

// Getters e Setters

}

```

### Passo 5: Criar o Repositório

Crie um repositório que extende `JpaRepository` para acessar o banco de dados. Exemplo: `src/main/java/com/example/crudapp/repository/ProdutoRepository.java`.

```java

package com.example.crudapp.repository;

import com.example.crudapp.model.Produto;

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

public interface ProdutoRepository extends JpaRepository<Produto, Long> {

}

```

### Passo 6: Criar o Serviço

Crie uma camada de serviço para a lógica de negócios. Exemplo: `src/main/java/com/example/crudapp/service/ProdutoService.java`.

```java

package com.example.crudapp.service;

import com.example.crudapp.model.Produto;

import com.example.crudapp.repository.ProdutoRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProdutoService {

@Autowired

private ProdutoRepository produtoRepository;

public List<Produto> listarTodos() {

return produtoRepository.findAll();

}

public Produto salvar(Produto produto) {

return produtoRepository.save(produto);

}

public Produto buscarPorId(Long id) {

return produtoRepository.findById(id).orElse(null);

}

public void excluir(Long id) {

produtoRepository.deleteById(id);

}

}

```

### Passo 7: Criar o Controlador

Implemente um controlador REST para o CRUD. Exemplo: `src/main/java/com/example/crudapp/controller/ProdutoController.java`.

```java

package com.example.crudapp.controller;

import com.example.crudapp.model.Produto;

import com.example.crudapp.service.ProdutoService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/produtos")

public class ProdutoController {

@Autowired

private ProdutoService produtoService;

@GetMapping

public List<Produto> listarTodos() {

return produtoService.listarTodos();

}

@PostMapping

public Produto salvar(@RequestBody Produto produto) {

return produtoService.salvar(produto);

}

@GetMapping("/{id}")

public ResponseEntity<Produto> buscarPorId(@PathVariable Long id) {

Produto produto = produtoService.buscarPorId(id);

if (produto != null) {

return ResponseEntity.ok(produto);

} else {

return ResponseEntity.notFound().build();

}

}

@PutMapping("/{id}")

public ResponseEntity<Produto> atualizar(@PathVariable Long id, @RequestBody Produto produto) {

Produto produtoExistente = produtoService.buscarPorId(id);

if (produtoExistente != null) {

produtoExistente.setNome(produto.getNome());

produtoExistente.setPreco(produto.getPreco());

produtoService.salvar(produtoExistente);

return ResponseEntity.ok(produtoExistente);

} else {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> excluir(@PathVariable Long id) {

Produto produto = produtoService.buscarPorId(id);

if (produto != null) {

produtoService.excluir(id);

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

```

### Passo 8: Testar a Aplicação

- Rode o projeto no IntelliJ IDEA.

- Utilize o \*\*Postman\*\* ou o \*\*cURL\*\* para testar os endpoints:

- \*\*GET\*\*: `http://localhost:8080/produtos` (Lista todos os produtos)

- \*\*POST\*\*: `http://localhost:8080/produtos` (Cria um novo produto)

- \*\*GET\*\*: `http://localhost:8080/produtos/{id}` (Obtém um produto por ID)

- \*\*PUT\*\*: `http://localhost:8080/produtos/{id}` (Atualiza um produto)

- \*\*DELETE\*\*: `http://localhost:8080/produtos/{id}` (Exclui um produto)

Com esses passos, você terá uma aplicação CRUD simples utilizando Spring Boot, PostgreSQL e IntelliJ IDEA.