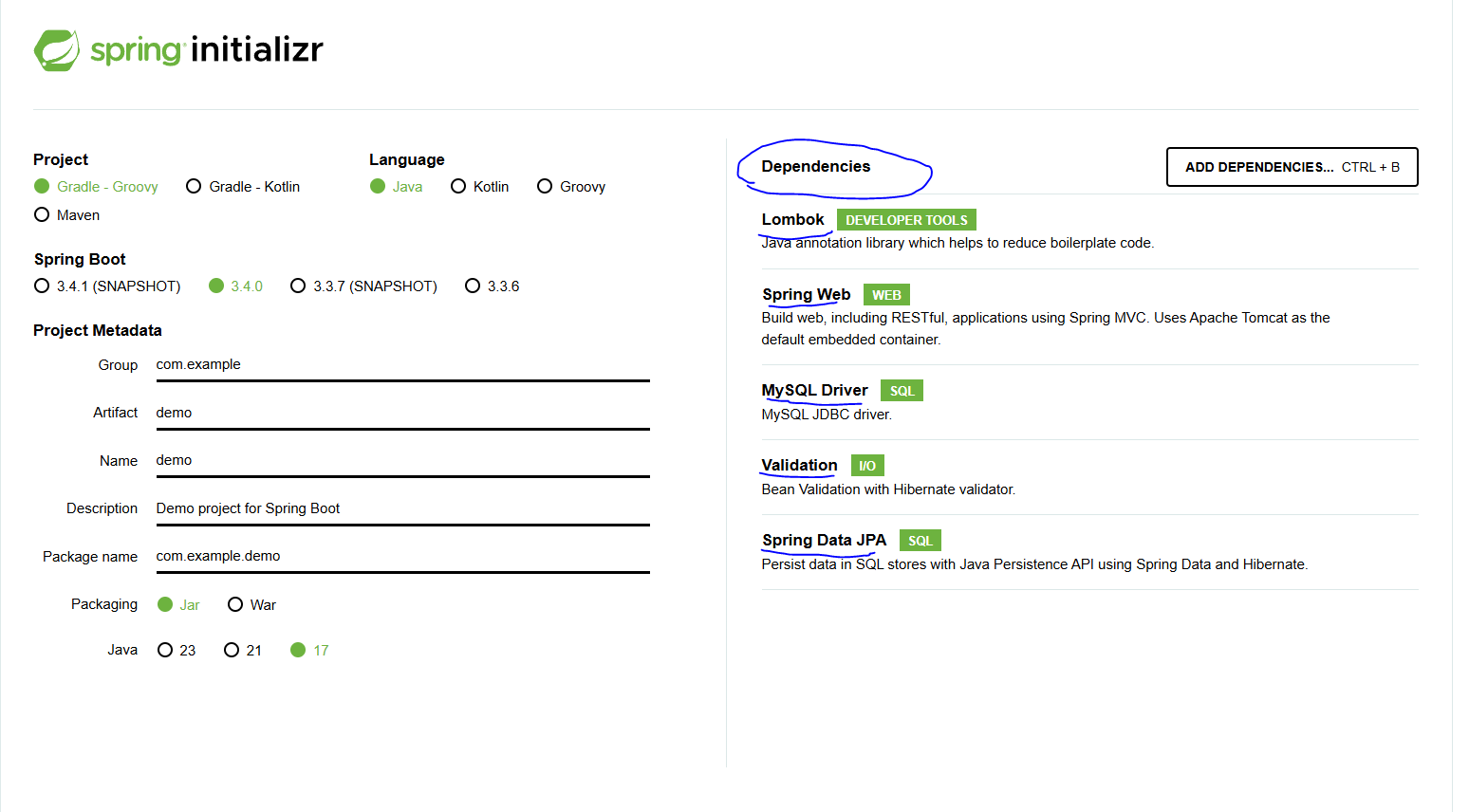
Documentação do crud em Java

## Primeiro passo, acessei o Spring boot e coloquei as dependências necessárias, exemplos:

* Lombok
* Spring Data JPA
* Spring Web
* Validation



Coloquei a pasta que eu gerei dentro do eclipse e

comecei codando pelo dto

Linha 1.

#### package com.example.Moda.dto;

Essa linha define o package em que a classe ou a estrutura está localizada.

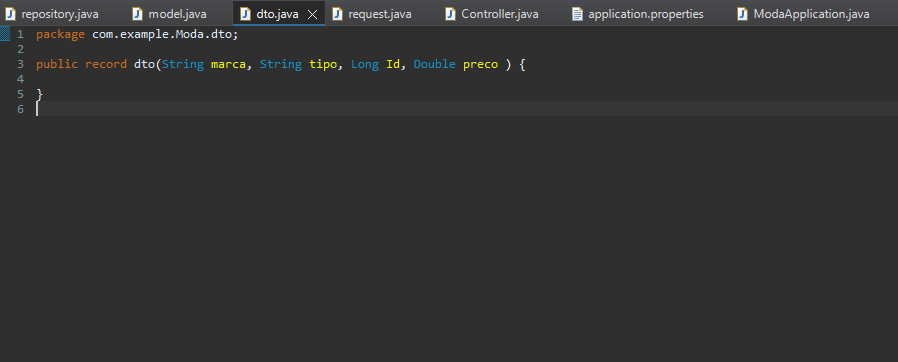
Linha 2

#### public record dto(String marca, String tipo, Long Id, Double preco ) {

Essa linha define um **record** chamado dto.

* A parte String marca, String tipo, Long Id, Double preco define **campos** para o *record*.

A estrutura de dados que você criou, um **record** em Java, é usada para representar um **objeto imutável** que contém **dados** de forma compacta e fácil de usar



**Prosseguindo criei o repository**

#### package com.example.Moda.repository;

Essa linha define o **pacote** onde essa interface está localizada.

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

Esta linha importa a interface JpaRepository do Spring Data JPA.

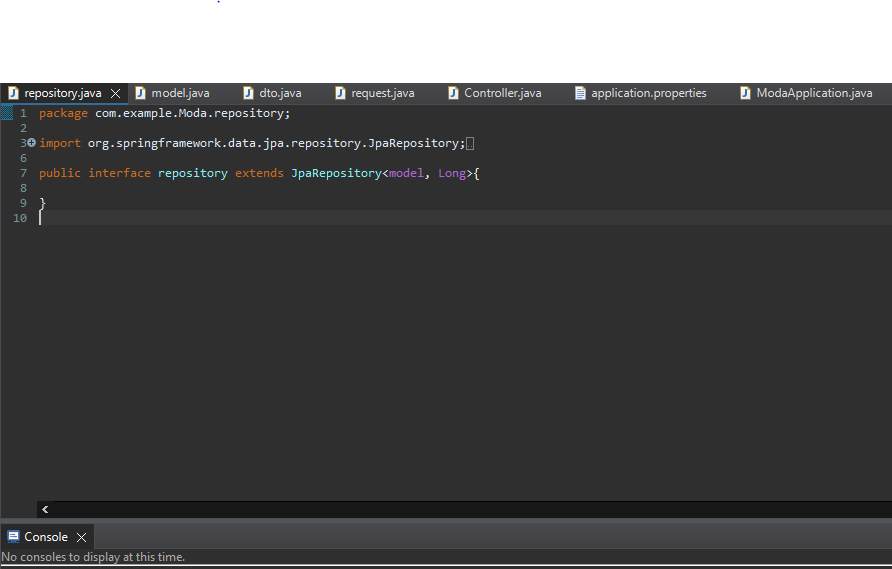
import com.example.Moda.model.model;

Essa linha importa a classe model que está localizada no pacote com.example.Moda.model.

public interface repository extends JpaRepository<model, Long> {

Esta linha define uma **interface** chamada repository que **extende** a interface JpaRepository.

Resumo do que a interface repository faz:

Ela herda os métodos prontos do JpaRepository para realizar operações no banco de dados (CRUD). 

Depois partir e criei a model

#### 1 package com.example.Moda.model;

Esta linha define o **pacote** onde essa classe está localizada.

#### 2import com.example.Moda.dto.dto;

Essa linha importa a classe dto que está no pacote com.example.Moda.dto.

#### 3import jakarta.persistence.Entity;

#### **4.** import jakarta.persistence.GeneratedValue;

#### **5.** import jakarta.persistence.GenerationType;

#### **6.** import jakarta.persistence.Id;

#### **7.** import jakarta.persistence.Table;

Estas linhas importam as anotações do **Jakarta Persistence API (JPA)**. JPA é a especificação que o Java usa para interagir com bancos de dados relacionais de maneira orientada a objetos.

@Entity: Indica que a classe é uma **entidade JPA**, ou seja, ela representa uma tabela no banco de dados.

@Table(name = "fashion"): Define o nome da tabela que a entidade model representa no banco de dados. Nesse caso, a tabela será chamada de "fashion".

@Id: Indica que o campo id é a chave primária da tabela.

@GeneratedValue(strategy = GenerationType.IDENTITY): Define que o valor do campo id será gerado automaticamente pelo banco de dados (no estilo de incremento automático de chave primária).

#### 8.import lombok.AllArgsConstructor;

#### **9.** import lombok.Getter;

#### **10.** import lombok.NoArgsConstructor;

#### **11.** import lombok.Setter;

Essas linhas importam as anotações do **Lombok**, uma biblioteca que reduz a quantidade de código repetitivo no Java.

@Getter: Gera os métodos getter para todos os campos da classe, ou seja, métodos como getId(), getTipo(), etc.

@Setter: Gera os métodos setter para todos os campos da classe, ou seja, métodos como setId(), setTipo(), etc.

@NoArgsConstructor: Gera o construtor padrão (sem argumentos) da classe.

@AllArgsConstructor: Gera um construtor que recebe todos os campos da classe como parâmetros, permitindo criar objetos passando todos os valores ao mesmo tempo.

#### public class model {

Aqui é onde a **classe** model é definida. Essa classe é uma entidade JPA que representa um **produto** no sistema de moda.

#### private Long id;

#### private String tipo;

#### private String marca;

#### private Double preco;

Esses são os **atributos** da classe model. Eles representam os dados de um produto na tabela fashion:

id: A chave primária única do produto (gerada automaticamente).

tipo: O tipo do produto (ex: "Tênis", "Camisa").

marca: A marca do produto (ex: "Nike", "Adidas").

preco: O preço do produto.

Esses campos serão mapeados para as colunas da tabela fashion no banco de dado

#### public model(dto request) {

Este é um **construtor** personalizado que recebe um objeto do tipo dto como parâmetro. Ele é usado para criar um objeto model a partir de um objeto dto

#### this.id = request.Id();

#### this.marca = request.marca();

#### this.preco = request.preco();

#### this.tipo = request.tipo();

Essas linhas atribuem os valores do DTO (passado como request) aos campos correspondentes da entidade model.

A classe model é uma entidade JPA, representando uma tabela no banco de dados chamada fashion. Cada instância da classe model corresponde a uma linha dessa tabela.

# 

Criei o request

record request(String marca, String tipo, Long Id, Double preco):

Este record é uma estrutura de dados imutável que contém 4 campos: marca, tipo, Id (provavelmente o identificador único do produto) e preco (o preço do produto).

Os fields são definidos como:

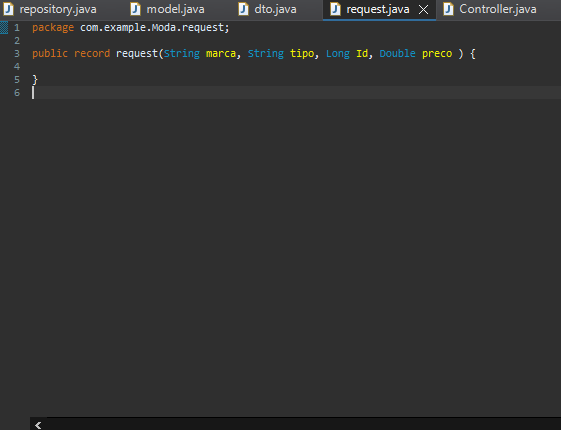
marca: A marca do produto (tipo String).

tipo: O tipo do produto (tipo String).

Id: O identificador do produto (tipo Long).

preco: O preço do produto (tipo Double).

Um *record* em Java é uma forma concisa de criar classes que servem apenas para armazenar dados, com alguns métodos essenciais (como toString(), equals(), hashCode()) gerados automaticamente.



Por último criei o Controller

**Anotações de Configuração do Spring:**

@RestController: Indica que a classe é um controlador **REST**, ou seja, ela irá lidar com as requisições HTTP e retornar respostas diretamente no formato JSON ou XML.

@RequestMapping("moda"): Define a URL base para os endpoints dessa classe como /moda.

**2. Dependência do Repositório:**

@Autowired repository repo;: Injeta automaticamente o **repositório** repository (provavelmente uma interface JPA) para acesso ao banco de dados e realizar operações CRUD.

**3. Métodos do Controller:**

@GetMapping:

**Função**: Recupera todos os produtos de moda do banco de dados.

**Implementação**: Chama repo.findAll() para obter uma lista de objetos model e retorna uma resposta HTTP 200 (ResponseEntity.ok(Lista)).

@PostMapping:

**Função**: Cria um novo produto de moda no banco de dados.

**Implementação**: Recebe um objeto DTO (fashion) via corpo da requisição, converte-o para um objeto model, e o salva no banco de dados através de repo.save(newmoda). Retorna uma resposta HTTP 200 após a inserção.

@PutMapping("/{id}"):

**Função**: Atualiza os dados de um produto existente no banco de dados.

**Implementação**: Busca um produto por id, se encontrado, atualiza os campos marca, tipo e preco com os dados do DTO (modadto) e salva as mudanças. Caso o produto não seja encontrado, retorna uma resposta HTTP 404 (ResponseEntity.notFound()).

@DeleteMapping("/{id}"):

**Função**: Exclui um produto do banco de dados.

**Implementação**: Recebe o id do produto e o deleta usando repo.deleteById(id), retornando uma resposta HTTP 204 (sem conteúdo).

**4. Transações e Validação:**

@Transactional: Garante que as operações de atualização e exclusão sejam feitas de forma transacional, ou seja, em um contexto de banco de dados seguro.

@Valid: Valida os dados recebidos via DTO antes de serem processados, garantindo que os valores atendam aos requisitos definidos (como tamanho, formato, etc.).

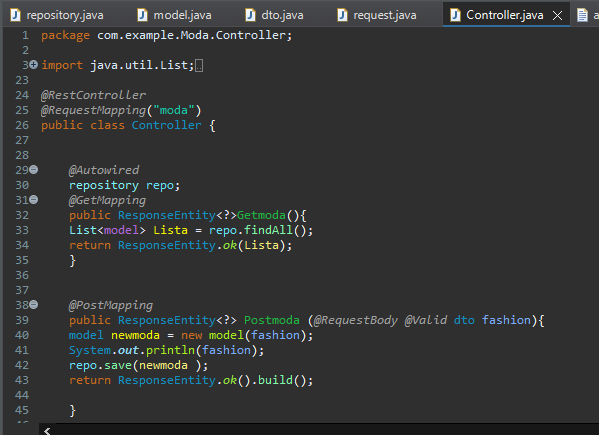
**Resumo da lógica dos métodos**:

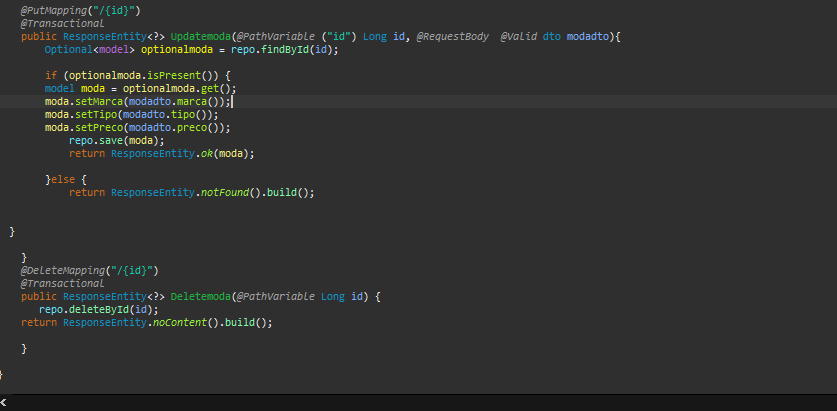
Getmoda(): Recupera e retorna todos os produtos cadastrados.

Postmoda(): Cria um novo produto a partir de um DTO.

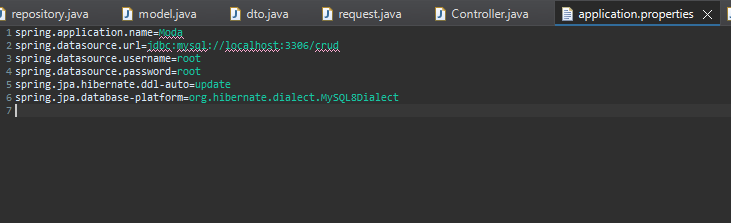
Updatemoda(): Atualiza um produto existente baseado no id, com dados enviados via DTO.

Deletemoda(): Exclui um produto com o id fornecido.

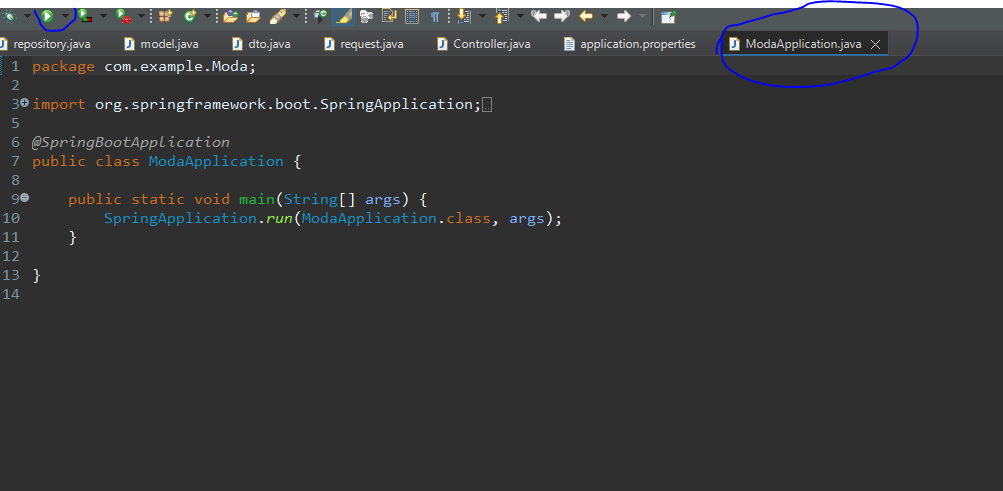


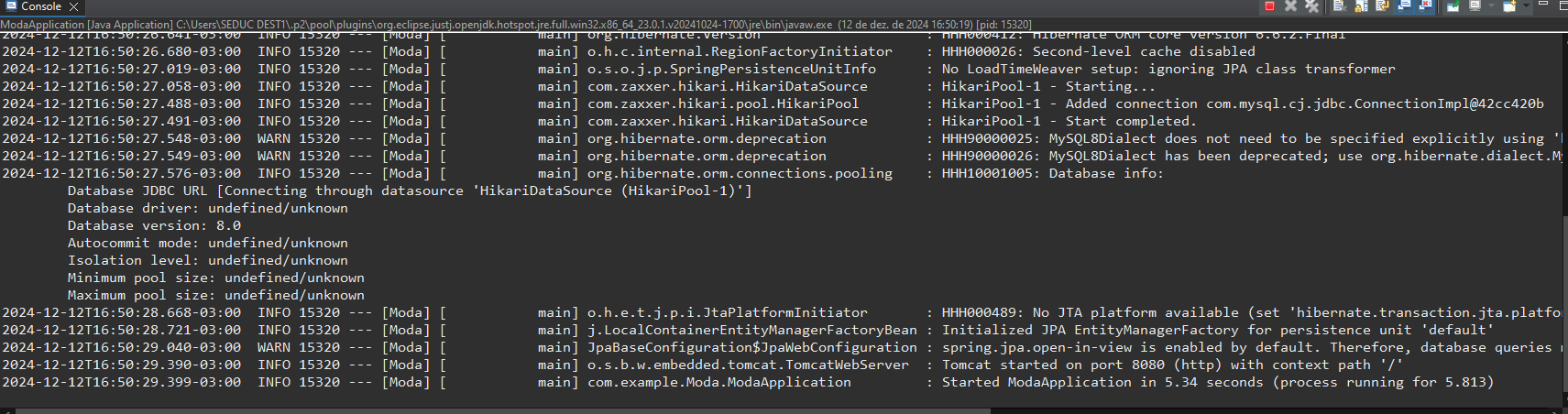


Coloquei as informações necessárias para conectar no banco, no aplication.propeties

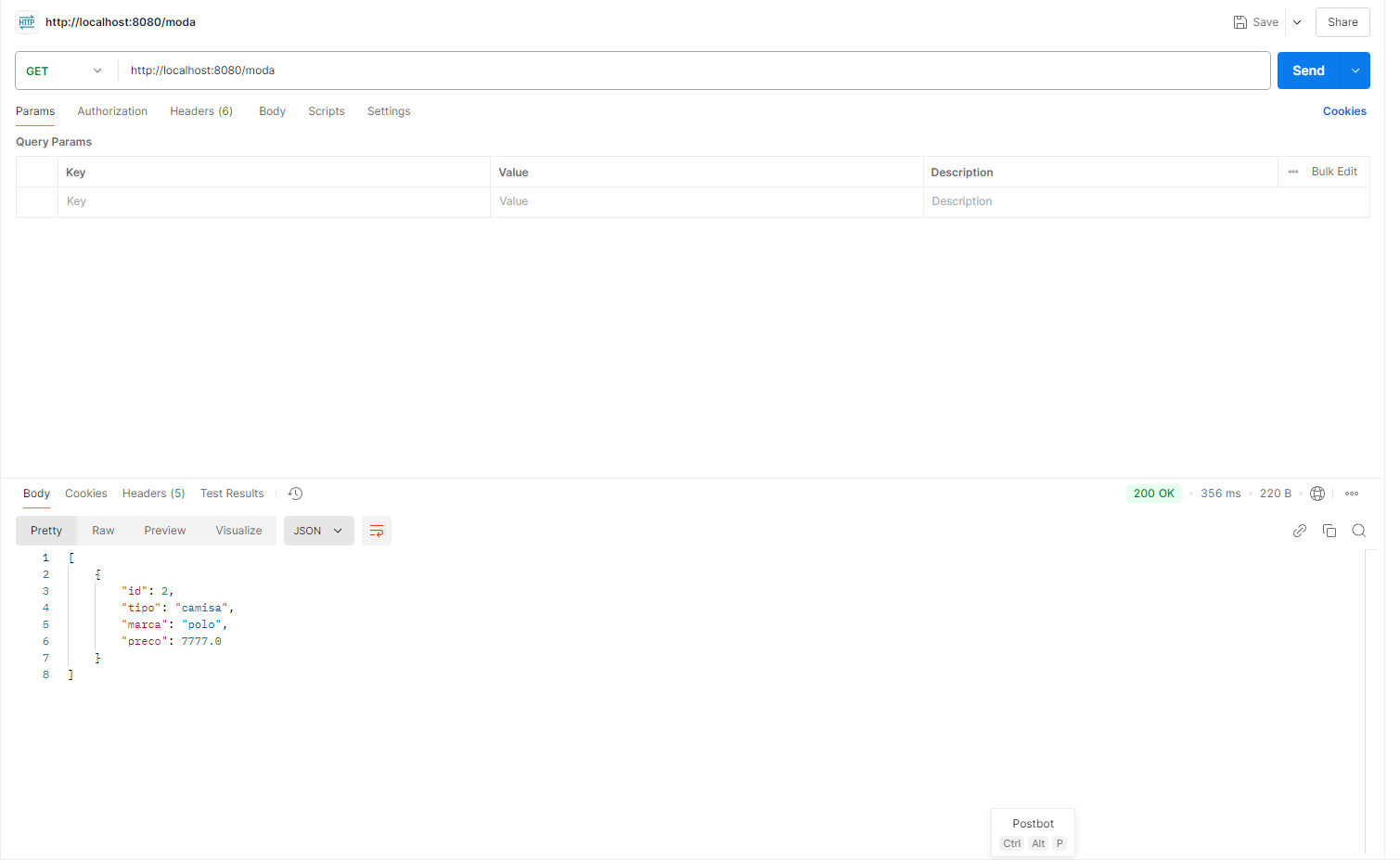


Por último rodei o código no ModaApplication que já vem gerado do spring boot



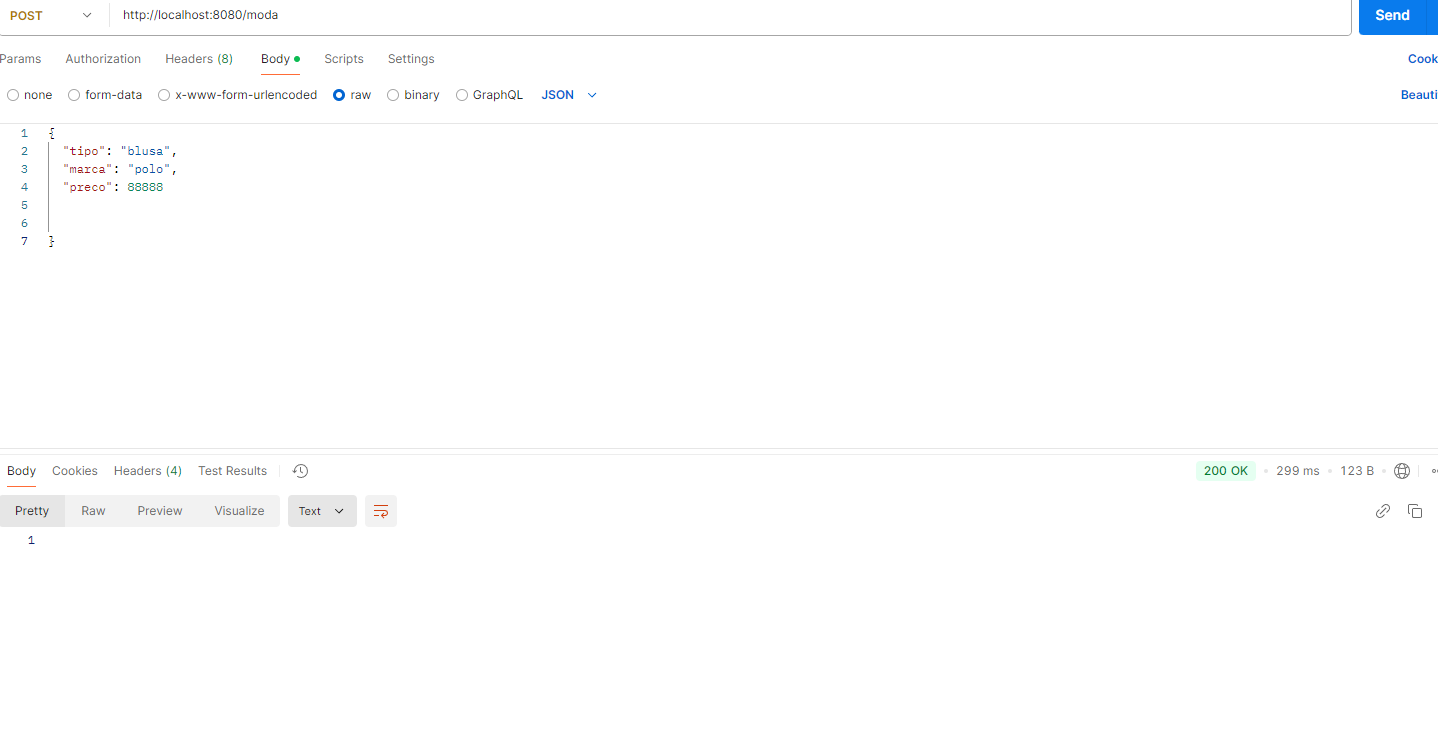
O código startou e fui testar no postaman 

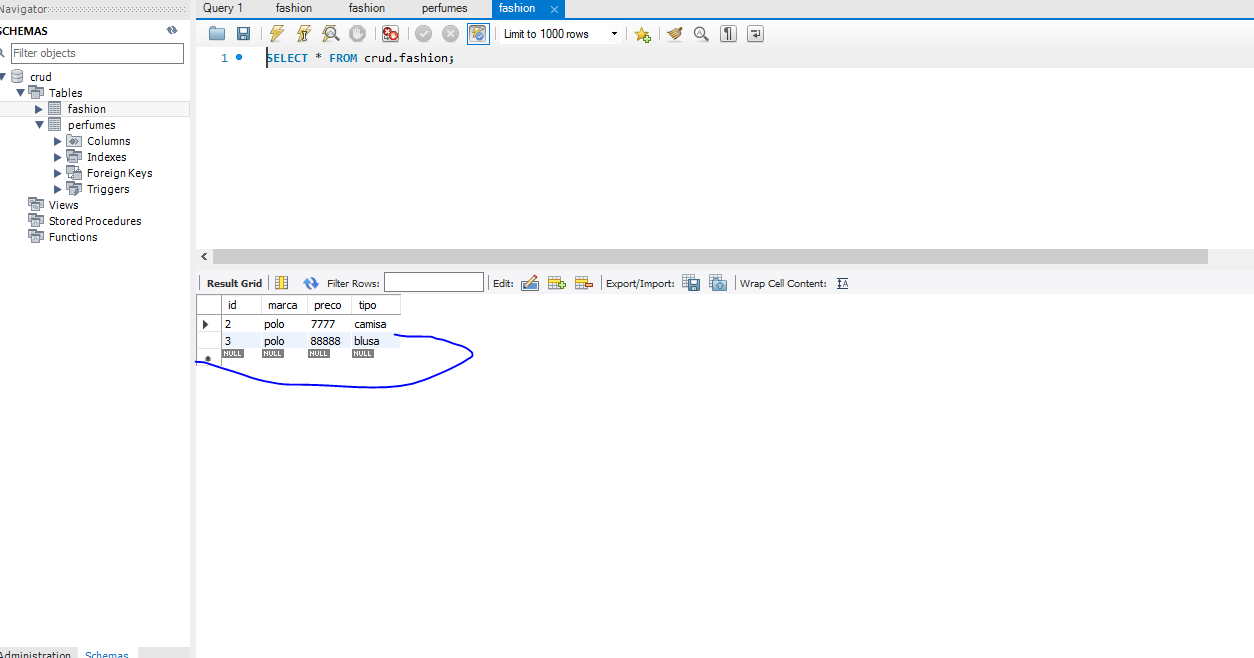
Testei o get



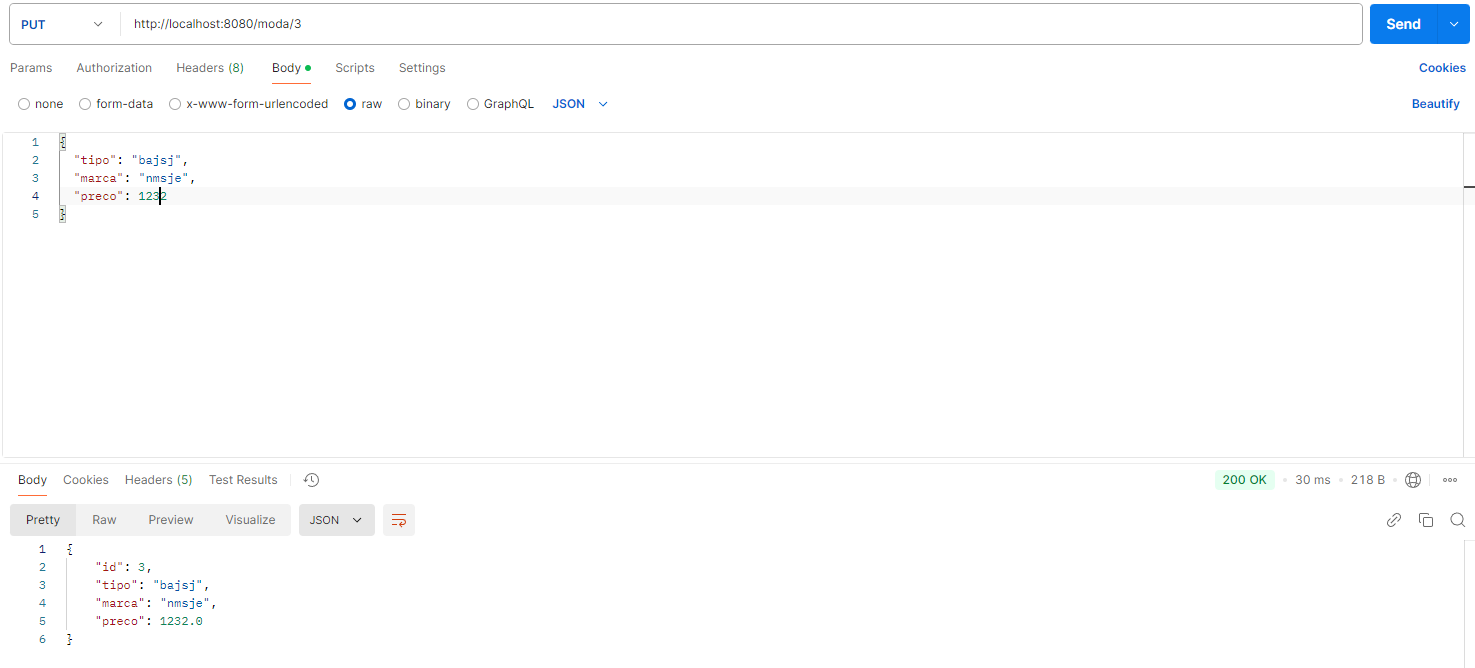
E foi no banco, e criou a tabela fashion.

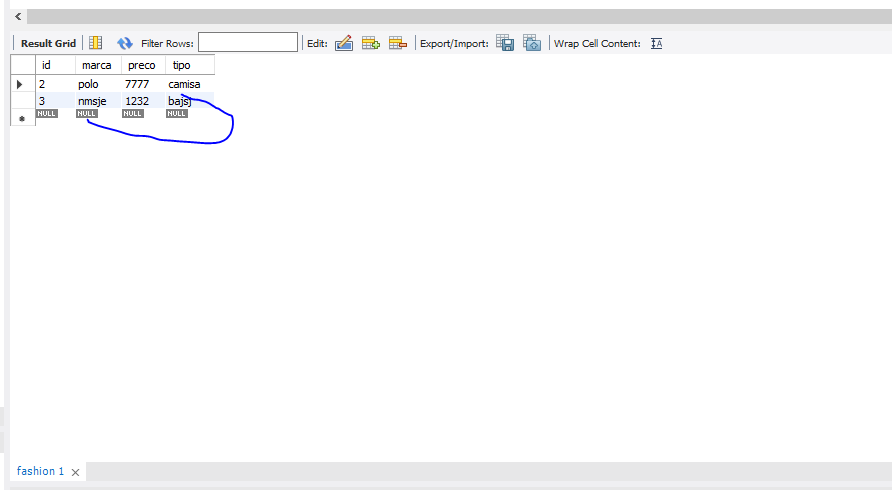
Testei o post e inserir os dados na tabela



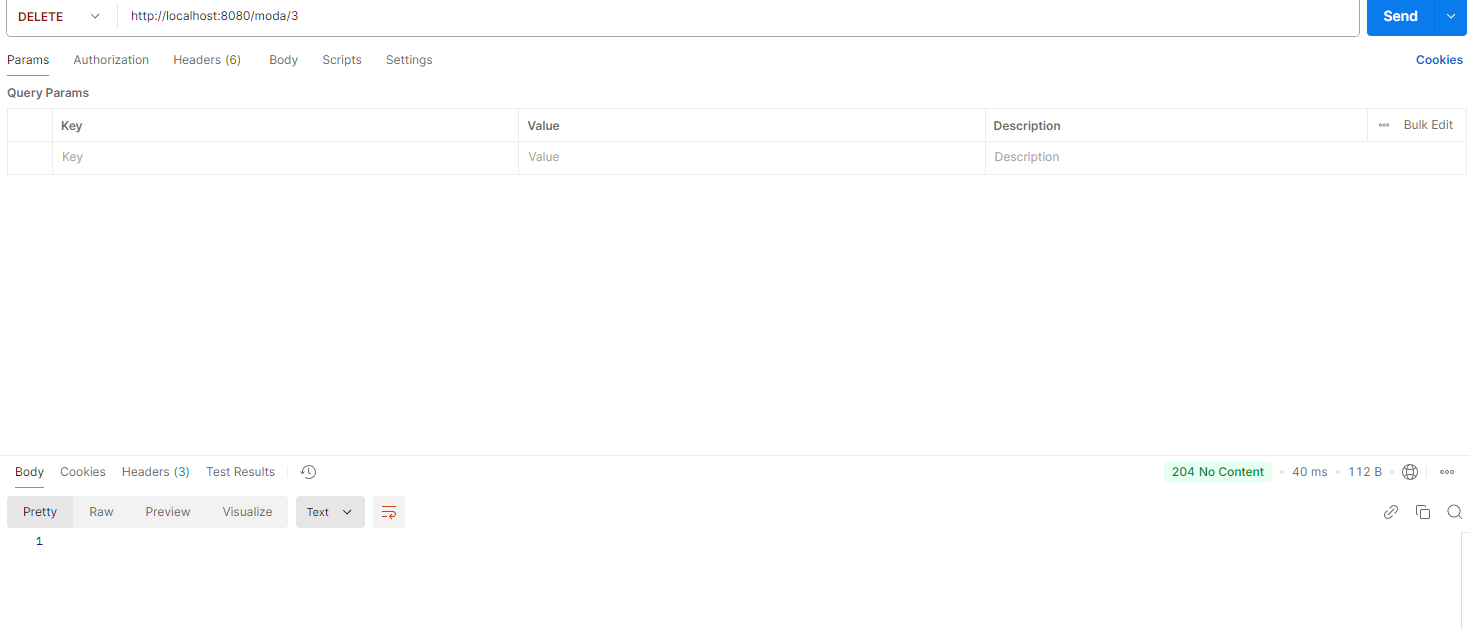


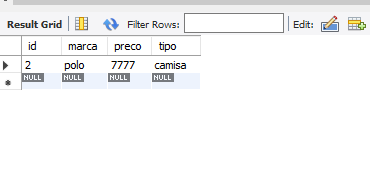
Depois atualizei os dados com o put





Por ultimo testei o delete e apaguei o id 3



e pronto

Ana Cariele!!!!