Bootcamp: Arquiteto de Software

Aluno(a): Paulo Gusttavo Tognato

Relatório de Entrega do Desafio Final

Introdução

O desafio final é uma atividade realizada pelo(a) aluno(a), **individualmente**, visando **levar à prática o conteúdo do bootcamp na totalidade**, utilizando as principais ferramentas, conteúdos, dinâmicas e modalidades orientadas pelo coordenador do curso.

O desafio final é **composto por um enunciado**, com viés de atividade prática (laboratório guiado, desafio a resolver, case a analisar etc.) com os principais tópicos vistos ao longo dos quatro módulos do bootcamp.

Quando necessário, o enunciado do desafio final pode ser acompanhado de uma videoaula gravada de orientação, onde o professor explica qual é a proposta do desafio final do bootcamp em questão, detalhando a dinâmica e elencando ferramentas/pré-requisitos para a resolução do enunciado.

Os principais passos, as escolhas e decisões tomadas para a resolução do enunciado do desafio final, bem como o resultado final, devem ser registrados pelo(a) aluno(a) neste documento, observando-se as **instruções de preenchimento**. Importante também observar os critérios no **protocolo avaliativo** da entrega.

Expirado o prazo de entrega, será disponibilizada uma **videoaula gravada contendo a resolução** da prática/problema/case proposto no enunciado do desafio final, bem como as devidas explicações acerca das decisões tomadas/ações executadas pelo professor para chegar no resultado.

Protocolo Avaliativo

O valor total do desafio final é de **95 pontos**. Ele será avaliado pelo professor do módulo em questão, com base no relatório do desafio final enviado pelo(a) aluno(a) e seguindo-se o protocolo avaliativo abaixo:

i. Todos os itens solicitados pelo(a) professor(a) no enunciado do desafio final foram entregues?

   Satisfatório - 19 pontos (50% a 100% dos itens)

   Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

   Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

   Insatisfatório/Não entregue - 0 pontos (0% dos itens)

ii. Os desafios/requisitos/funcionalidades foram resolvidos/considerados/implementados corretamente pelo(a) aluno(a)?

   Satisfatório - 19 pontos (50% a 100% dos itens)

   Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

   Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

   Insatisfatório/Não entregue - 0 pontos (0% dos itens)

iii. O(a) aluno(a) utilizou corretamente as ferramentas, conceitos e tecnologias apresentadas ao longo do bootcamp?

   Satisfatório - 19 pontos (50% a 100% dos itens)

   Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

   Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

iv. O(a) aluno(a) forneceu comentários detalhados e documentação completa, facilitando a compreensão da sua proposta de resolução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

v. O(a) aluno(a) entregou a solução com qualidade e zelo, demonstrando dedicação e empenho na solução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

Instruções de Preenchimento do Relatório do Desafio Final

O preenchimento do relatório deve ser realizado com base nas diretrizes abaixo, para assegurar uma avaliação justa e detalhada de cada item solicitado.

i. **Entrega Completa dos Itens Solicitados**

Verifique se todos os itens mencionados no enunciado do desafio final foram devidamente contemplados na sua entrega, pois a entrega deles é fundamental para a avaliação do desafio final na totalidade.

ii. **Resolução dos Desafios e Implementação dos Requisitos**

Avalie se você abordou corretamente o(s) desafio(s) proposto(s), considerando e implementando todas as funcionalidades e requisitos apresentados no enunciado. O objetivo é verificar se a solução atende às expectativas técnicas e funcionais solicitadas.

iii. **Utilização das Ferramentas, Conceitos e Tecnologias**

Certifique-se de que você aplicou corretamente as ferramentas, conceitos e tecnologias discutidas ao longo do bootcamp. A adequação e o uso eficaz dos recursos disponíveis são elementos essenciais para a solução do desafio.

iv. **Qualidade dos Comentários e Documentação**

Verifique se você forneceu os comentários necessários e detalhados acerca dos passos e decisões tomadas na resolução do desafio. Esses comentários servem para facilitar a compreensão da proposta de solução, demonstrando clareza e organização.

v. **Qualidade e Capricho na Entrega Final**

Observe se o seu relatório do desafio final apresenta um nível adequado de qualidade, zelo e capricho, de forma que evidencie sua dedicação, empenho e atenção aos detalhes durante o desenvolvimento da solução.

O preenchimento deve ser realizado com base em evidências concretas encontradas no trabalho entregue, assegurando que todos os aspectos mencionados acima sejam considerados.

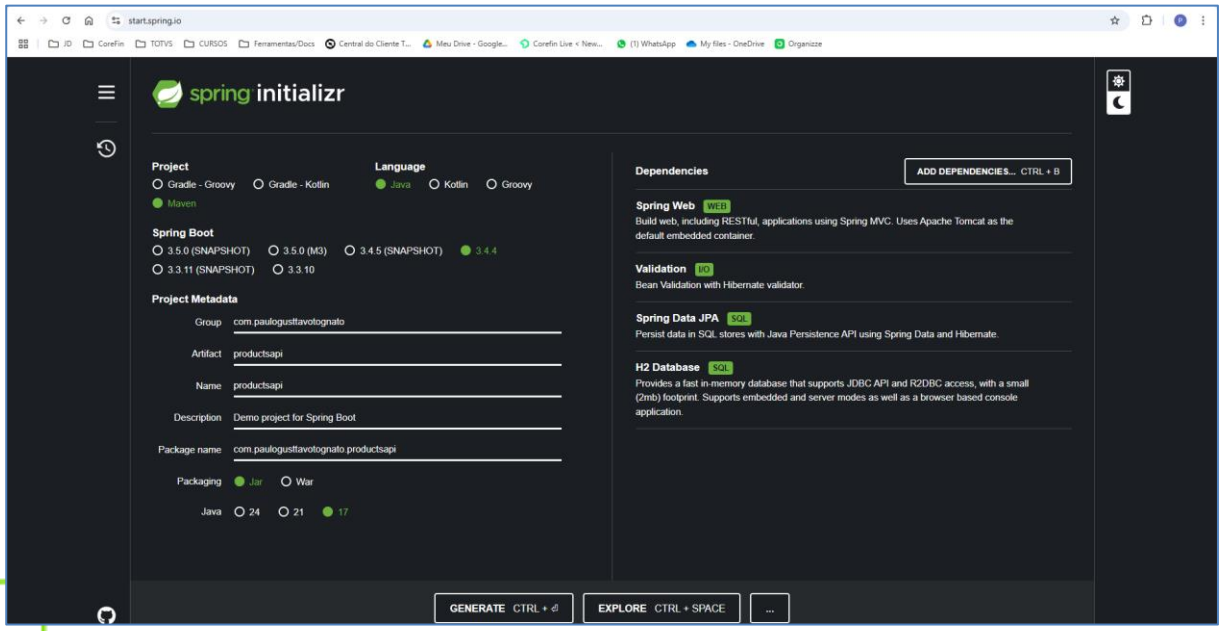
Desenvolvimento da Solução do Desafio Final

1. Projeto disponibilizado e documentado no Github.

Esse projeto foi documentado e disponibilizado no GITHUB e através da URL https://github.com/tognatopaulo/productapi é possível acessar e baixar o projeto.

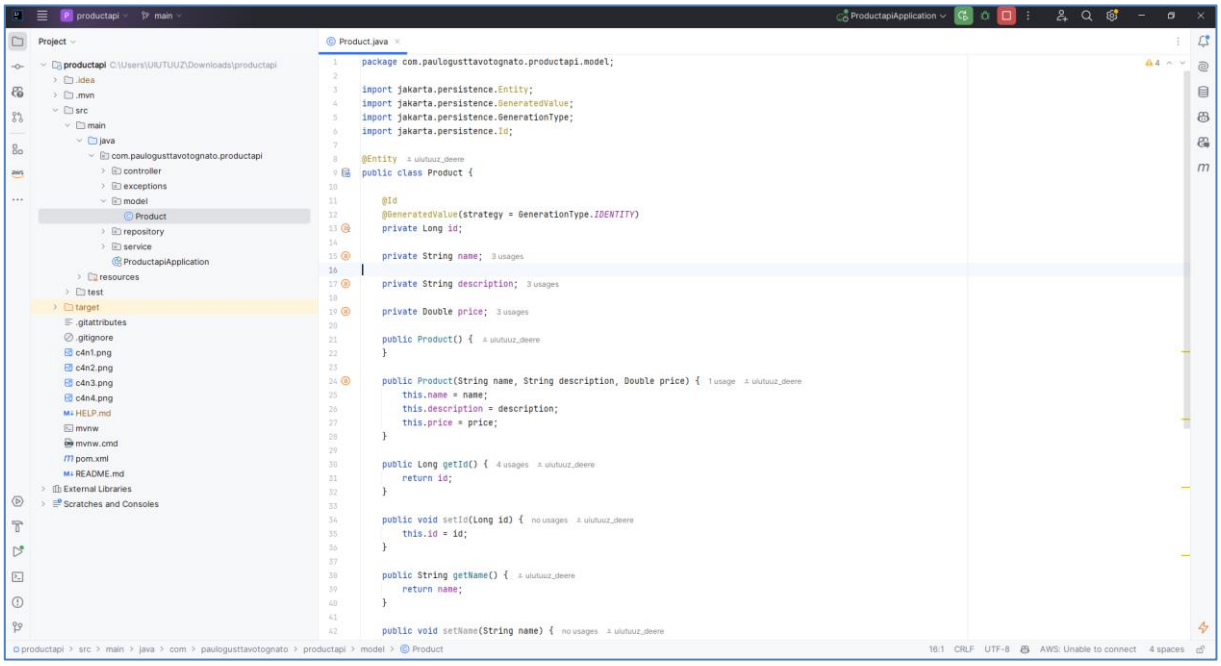
2. Criação do projeto usando o Spring Initializr.

Criado um projeto Java com Spring Boot com as seguintes dependências iniciais: Spring Web (classes para lidar com Rest API), Spring Data JPA (Classes para lidar com o mapeamento das entidades e banco de dados), H2 Database (Classes para criação e gerenciamento do banco de dados H2 em memória).



3. Criação do modelo de domínio PRODUCT.

Foi criado a entidade “Product” dentro do pacote Model com os campos: id, name, description e price.

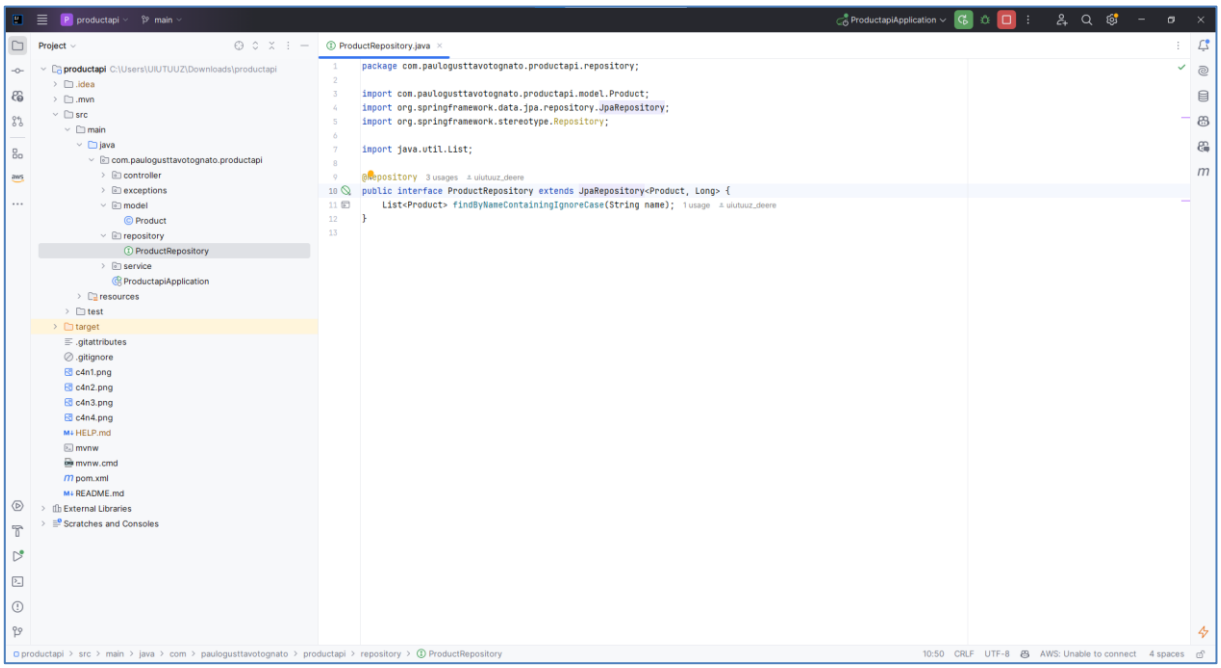


```java
package com.paulogusttavotognato.productapi.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Product {

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

    private String name;

    private String description;

    private Double price;

    public Product() {
    }

    public Product(String name, String description, Double price) {
        this.name = name;
        this.description = description;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
```

```
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }
}
```

4. Criação da interface de Repositório.

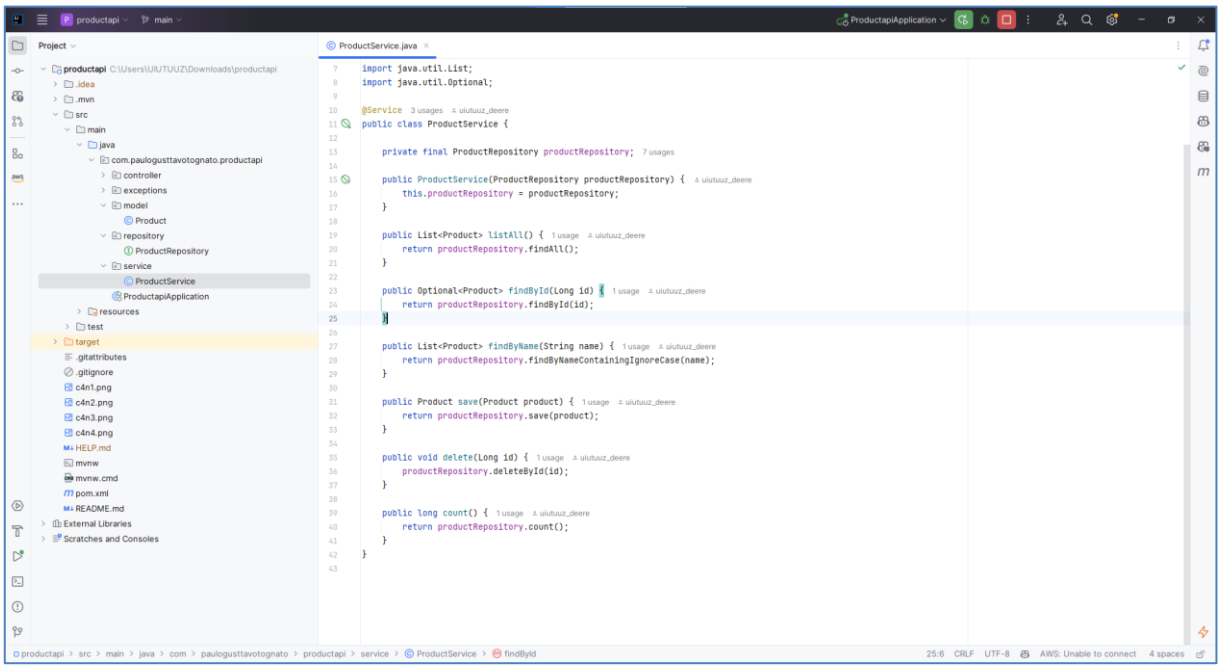
Foi criada a interface de repositório “ProductRepository” no pacote repositor. Essa interface estende a interface JpaRepository que é responsável pela integração da entidade com o banco de dados via JPA/Hibernate. Além disso, também foi necessário criar um método customizado para recuperar os produtos pelo seu nome.




```java
package com.paulogusttavotognato.productapi.repository;

import com.paulogusttavotognato.productapi.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNameContainingIgnoreCase(String name);
}
```

5. Criação da lógica de negócio na camada de serviço.

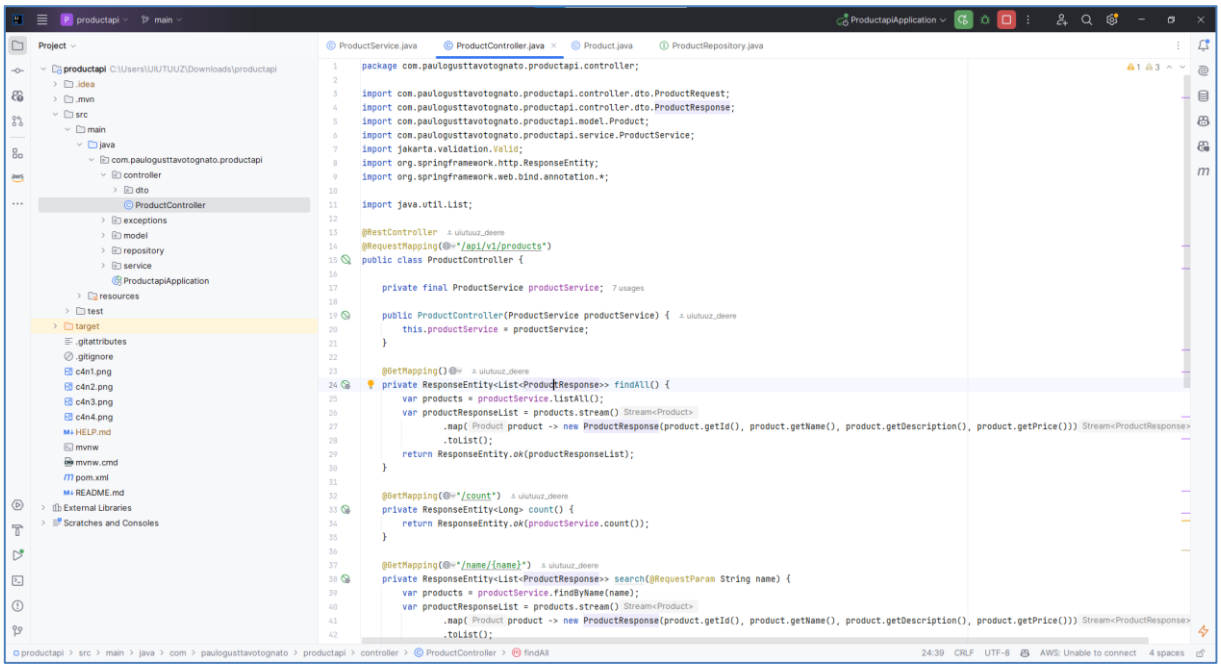
Criado a classe ProductService com todos os métodos CRUD e método de contagem solicitados no enunciado do projeto. Essa classe usa como dependência a classe de repositórios que é responsável por realizar as queries no banco de dados.



```java
package com.paulogusttavotognato.productapi.service;

import com.paulogusttavotognato.productapi.model.Product;
import com.paulogusttavotognato.productapi.repository.ProductRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    public List<Product> listAll() {
        return productRepository.findAll();
    }

    public Optional<Product> findById(Long id) {
        return productRepository.findById(id);
    }

    public List<Product> findByName(String name) {
        return productRepository.findByNameContainingIgnoreCase(name);
    }

    public Product save(Product product) {
        return productRepository.save(product);
    }

    public void delete(Long id) {
        productRepository.deleteById(id);
    }

    public long count() {
        return productRepository.count();
    }
}
```

6. Criação dos endpoints REST na classe de Controller.

Criado o ProductController que é a classe responsável por mapear e expor os endpoints RESTFul.

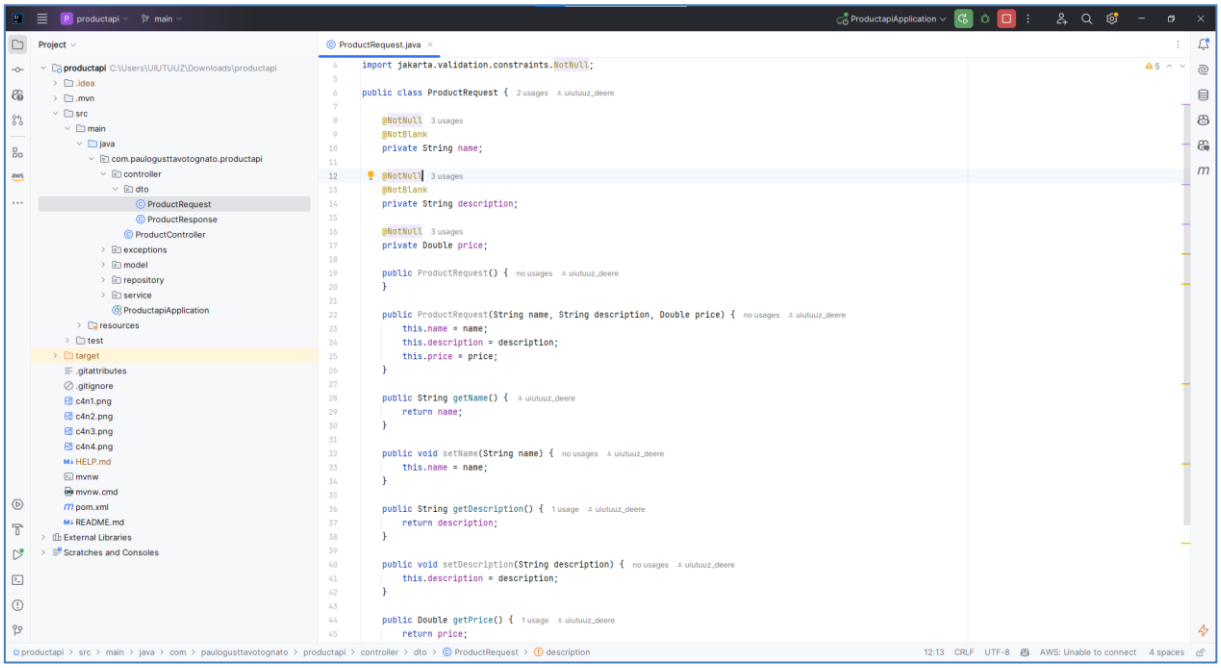


```java
package com.paulogusttavotognato.productapi.controller;

import com.paulogusttavotognato.productapi.controller.dto.ProductRequest;
import com.paulogusttavotognato.productapi.controller.dto.ProductResponse;
import com.paulogusttavotognato.productapi.model.Product;
import com.paulogusttavotognato.productapi.service.ProductService;
import jakarta.validation.Valid;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1/products")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    @GetMapping()
    private ResponseEntity<List<ProductResponse>> findAll() {
        var products = productService.listAll();
        var productResponseList = products.stream()
                .map(product -> new ProductResponse(product.getId(), product.getName(),
```

```
product.getDescription(), product.getPrice()))
                .toList();
        return ResponseEntity.ok(productResponseList);
    }

    @GetMapping("/count")
    private ResponseEntity<Long> count() {
        return ResponseEntity.ok(productService.count());
    }

    @GetMapping("/name/{name}")
    private ResponseEntity<List<ProductResponse>> search(@RequestParam String name) {
        var products = productService.findByName(name);
        var productResponseList = products.stream()
                .map(product -> new ProductResponse(product.getId(), product.getName(),
product.getDescription(), product.getPrice()))
                .toList();
        return ResponseEntity.ok(productResponseList);
    }

    @GetMapping("/{id}")
    private ResponseEntity<ProductResponse> findById(@RequestParam Long id) {
        var product = productService.findById(id);
        if (product.isPresent()) {
            var productResponse = new ProductResponse(product.get().getId(),
product.get().getName(), product.get().getDescription(), product.get().getPrice());
            return ResponseEntity.ok(productResponse);
        }
        return ResponseEntity.notFound().build();
    }

    @PostMapping()
    private ResponseEntity<ProductResponse> create(@Valid @RequestBody ProductRequest
product) {
        var newProduct = new Product(product.getName(), product.getDescription(),
product.getPrice());
        var savedProduct = productService.save(newProduct);
        var productResponse = new ProductResponse(savedProduct.getId(),
savedProduct.getName(), savedProduct.getDescription(), savedProduct.getPrice());
        return ResponseEntity.created(null).body(productResponse);
    }

    @DeleteMapping("/{id}")
    private ResponseEntity<Void> delete(@RequestParam Long id) {
        productService.delete(id);
        return ResponseEntity.noContent().build();
    }

}
```

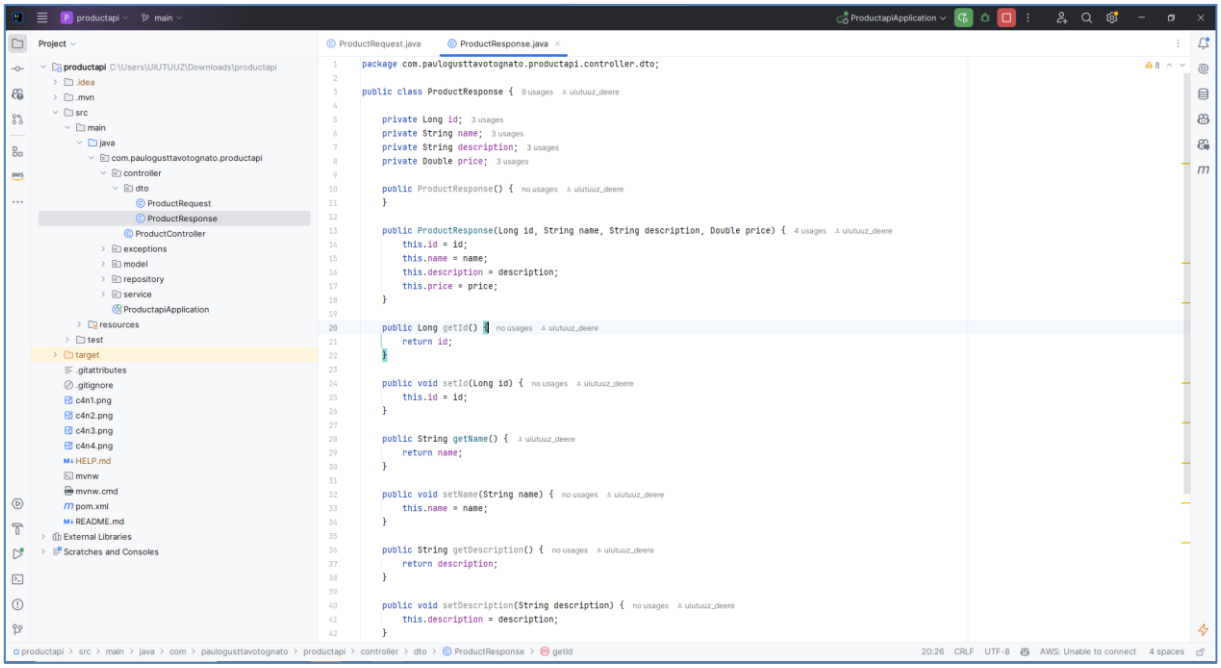
Além disso, como boas práticas, foram criadas 2 classes de DTO (Data Transfer Object) para lidar com os dados de requisição e de resposta da

aplicação.



```java
package com.paulogusttavotognato.productapi.controller.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

public class ProductRequest {

    @NotNull
    @NotBlank
    private String name;

    @NotNull
    @NotBlank
    private String description;

    @NotNull
    private Double price;

    public ProductRequest() {
    }

    public ProductRequest(String name, String description, Double price) {
        this.name = name;
        this.description = description;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
```

```
    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }
}
```

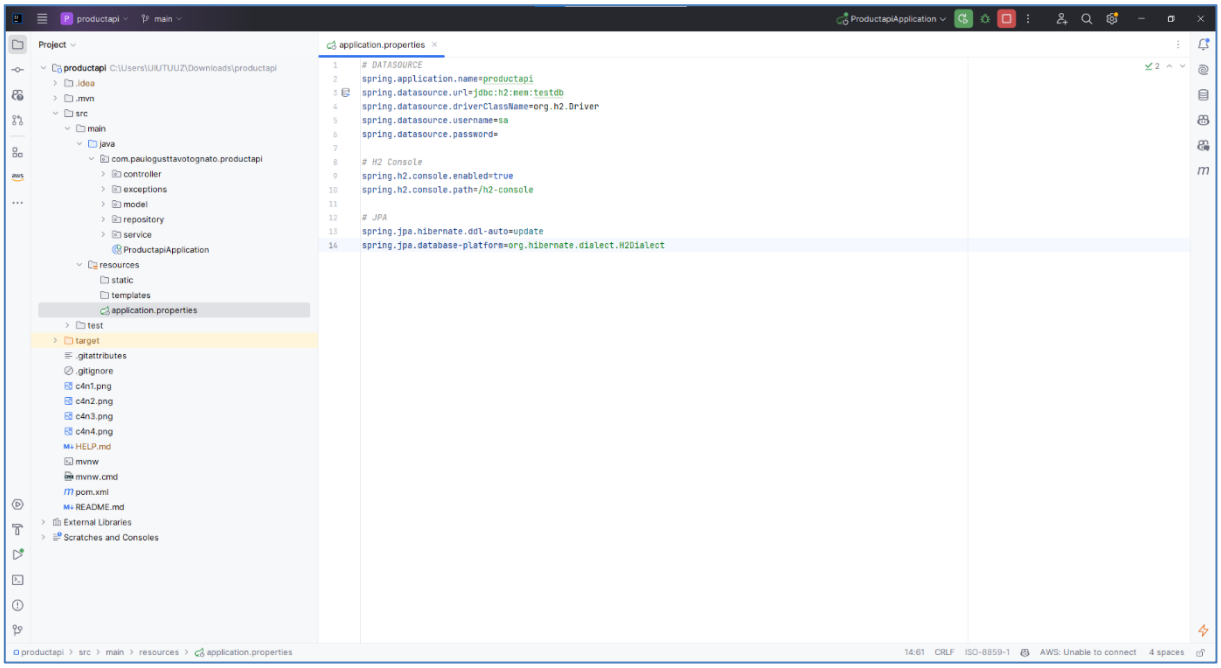



```
package com.paulogusttavotognato.productapi.controller.dto;

public class ProductResponse {

    private Long id;
    private String name;
    private String description;
    private Double price;

    public ProductResponse() {
    }

    public ProductResponse(Long id, String name, String description, Double price) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }
}
```

7. Configuração do banco de dados H2.

As propriedades da aplicação foram configuradas com as configurações do banco de dados H2 em memória e foi habilitado o console do H2.

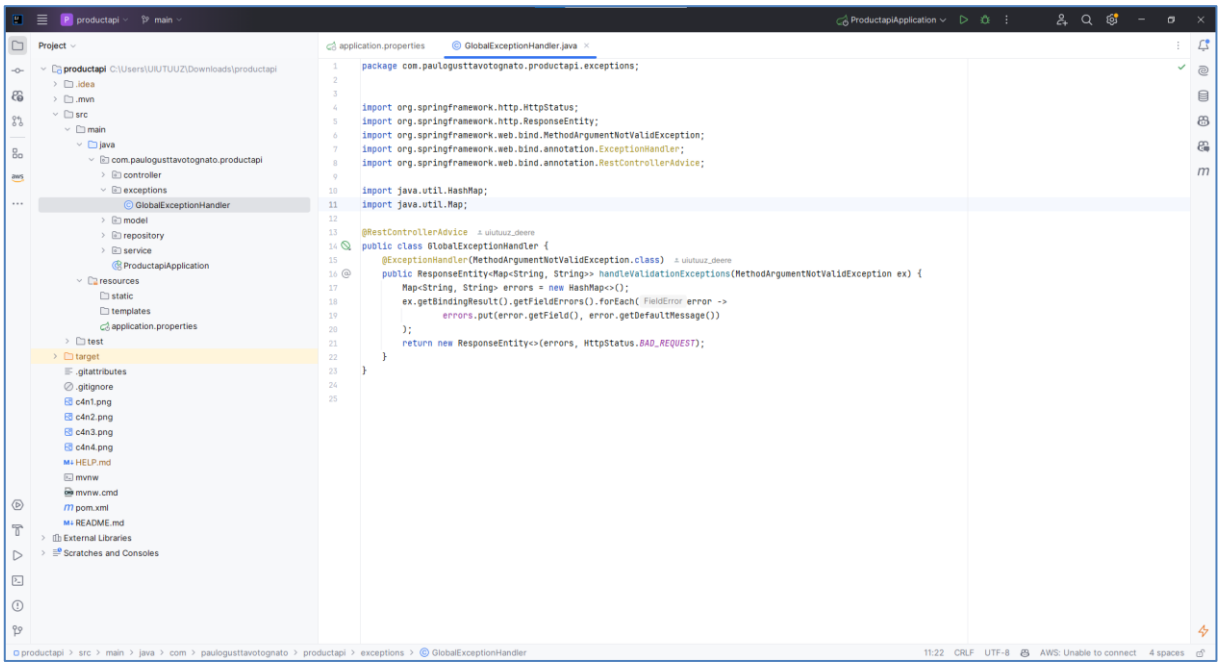



```
#DATASOURCE
spring.application.name=productapi
```

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

#H2 Console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

#JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

8. Criação de classe para controle de exceções.

Foi criado uma classe para controlar as exceções do programa. Além disso, foi implementado um método para controle das exceções das validações dos campos da requisição do método POST da aplicação, para garantir que os campos obrigatórios sejam sempre preenchidos.

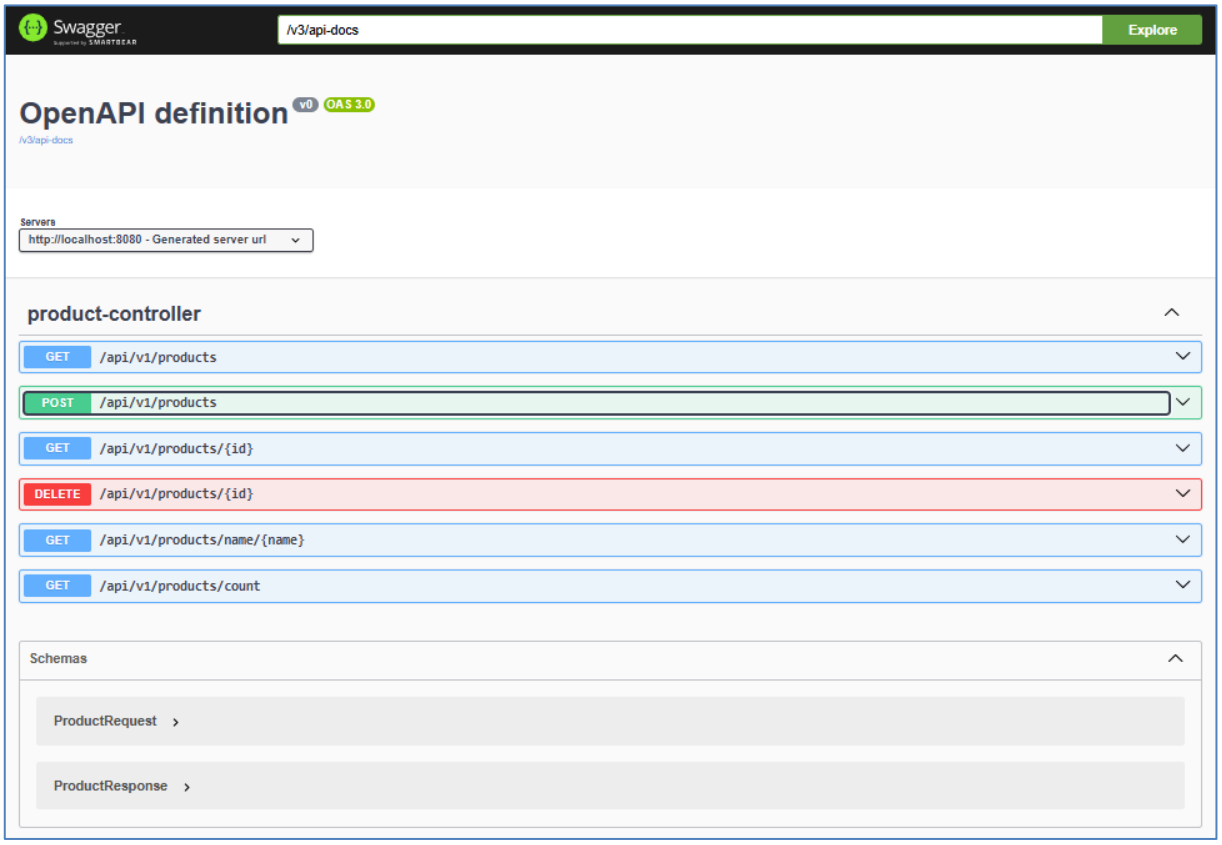



```
package com.paulogusttavotognato.productapi.exceptions;


import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {
```

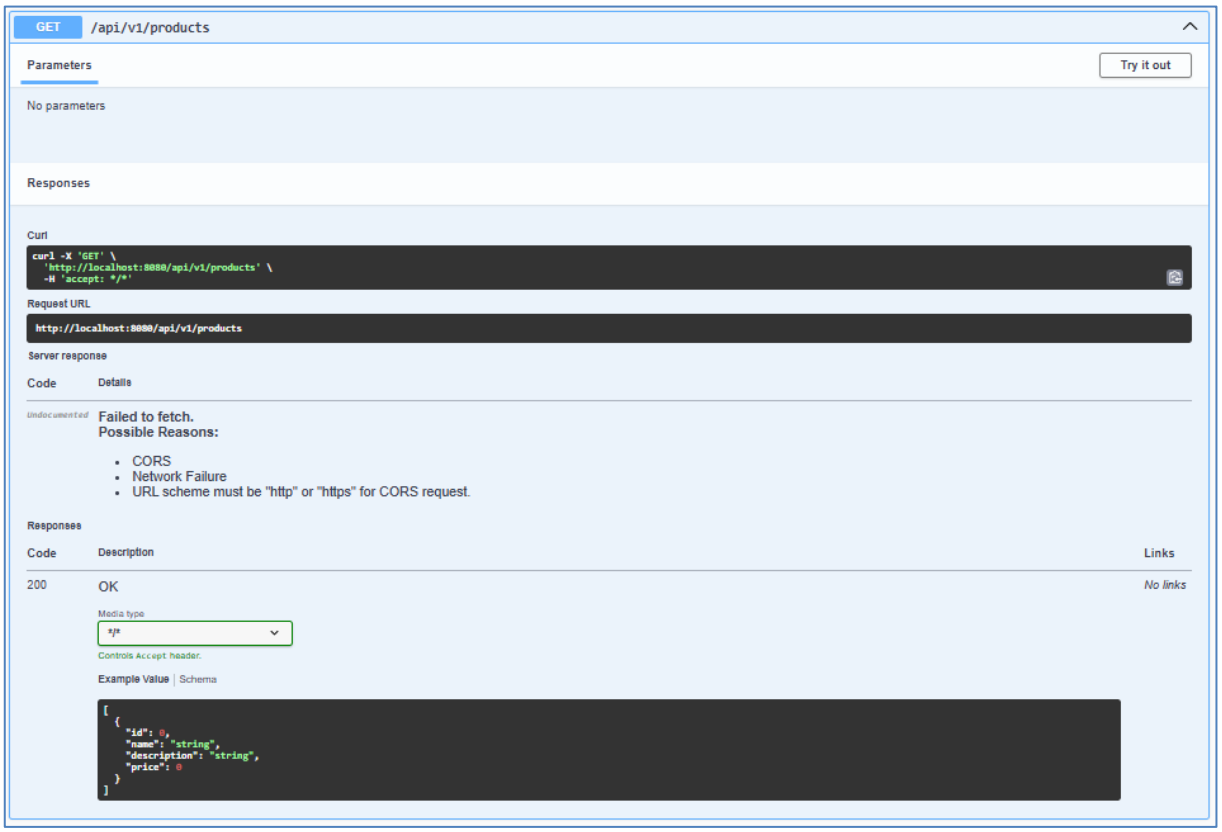
```
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>>
handleValidationExceptions(MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error ->
                errors.put(error.getField(), error.getDefaultMessage())
        );
        return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
    }
}
```

9. Implementação do Swagger e OpenAPI.

Foi implementando na aplicação a dependência do Swagger e OpenAPI para geração automática da documentação com base nos endpoints expostos pelo controller. Desta forma é possível ver a documentação a partir do endpoint "/swagger-ui/index.html" e também realizar as operações dentro dessa interface gráfica.

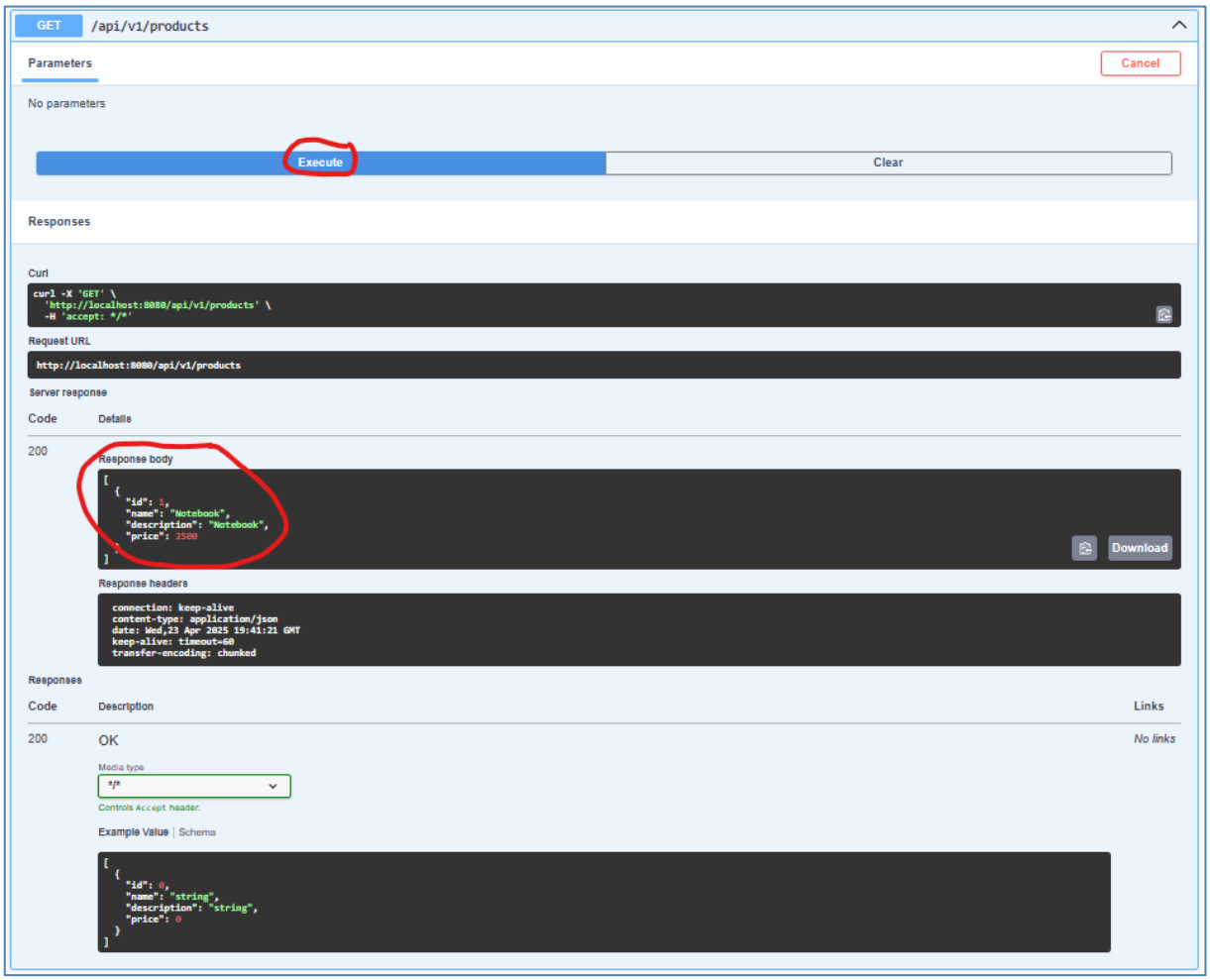


Ao clicar no endpoint exposto, é possível ver a sua documentação, bem como o exemplo de requisição e de resposta.

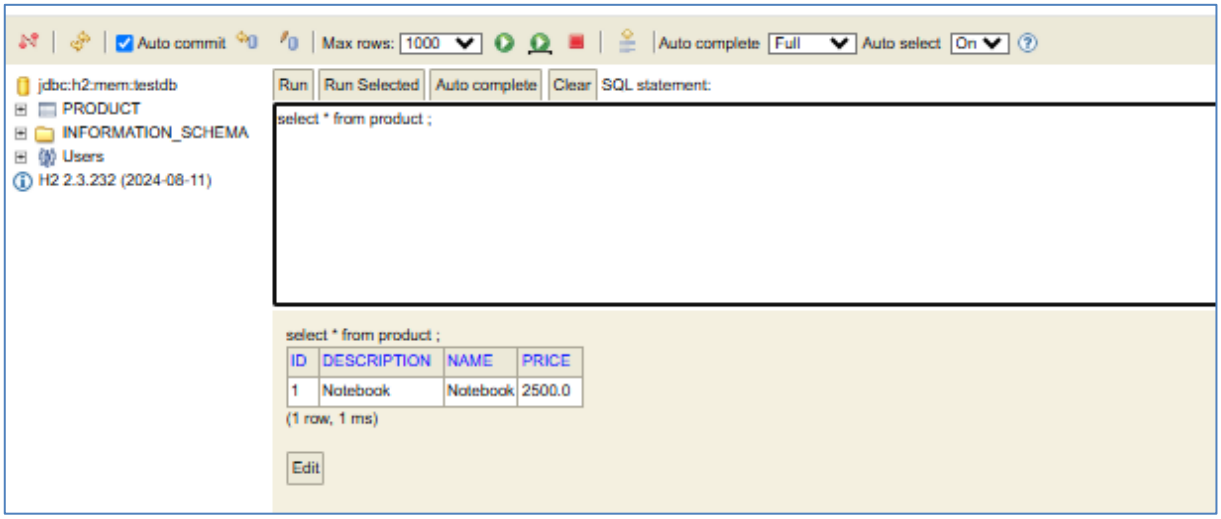


Também é possível ver o botão “Try it out”, na qual você pode clicar e a partir de então pode realizar as chamadas diretamente para a aplicação. Neste caso, ao clicar em “execute” podemos ver o retorno de um produto que já está cadastrado no banco de dados.

- **GET api/v1/products:**

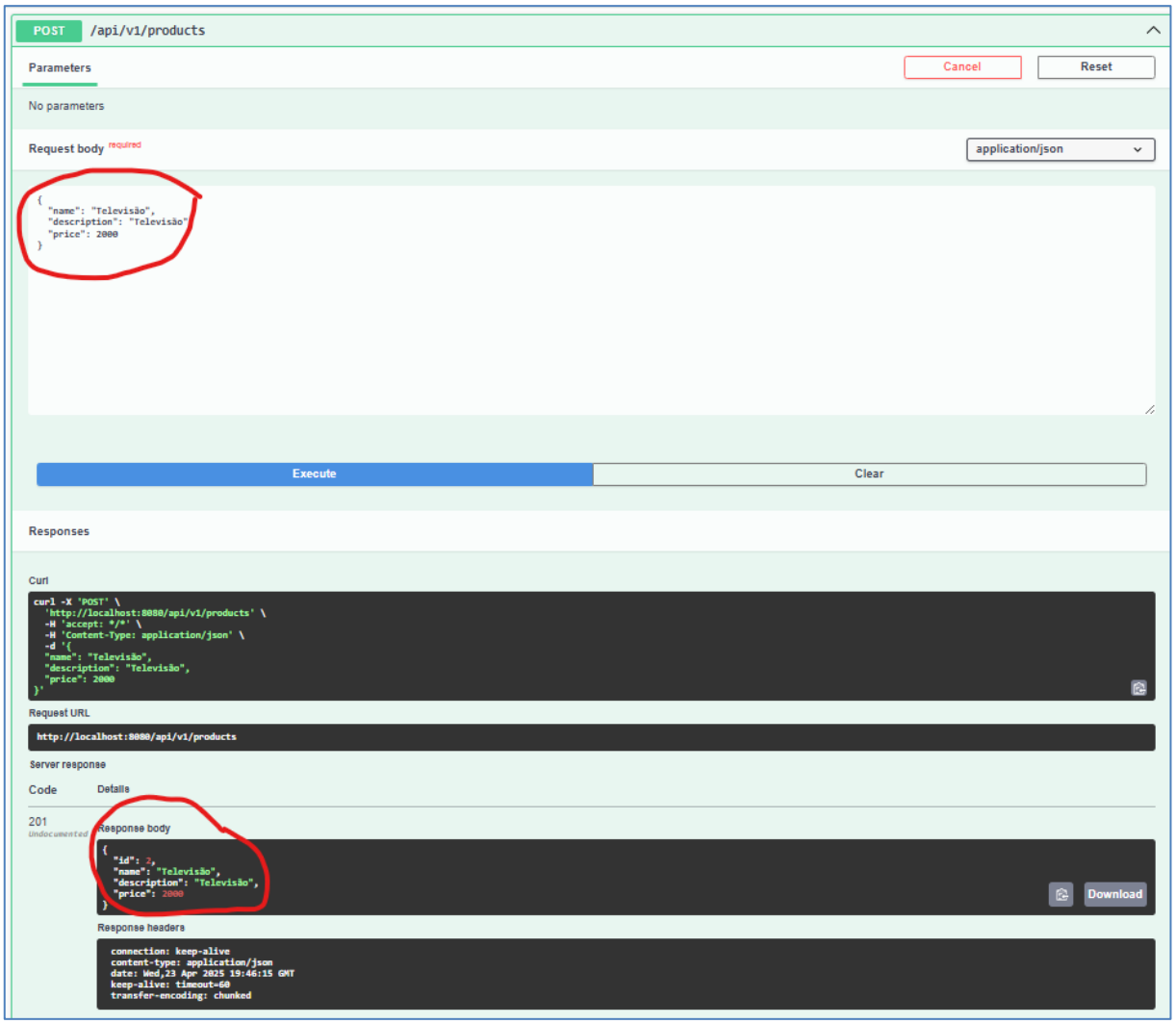


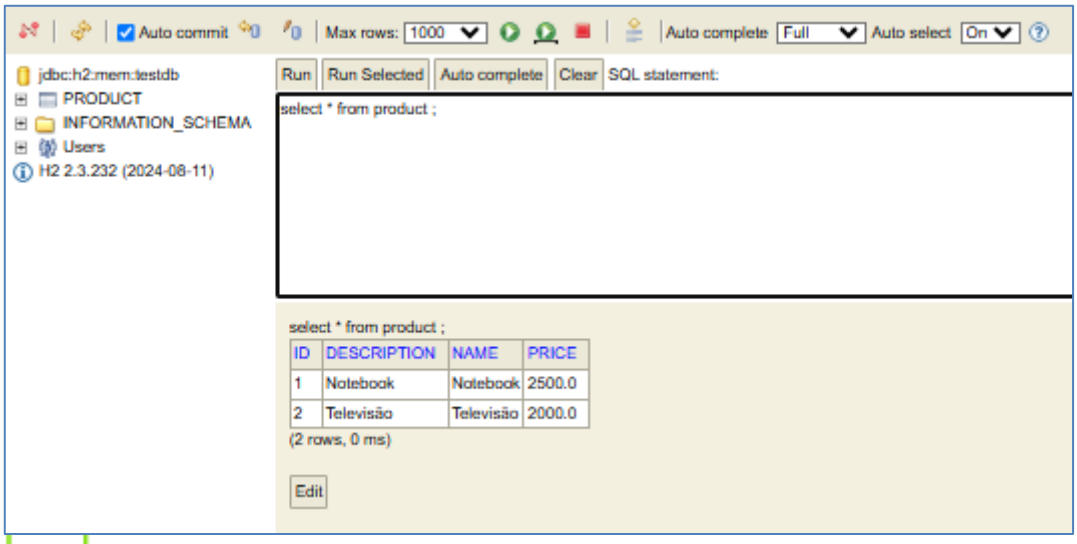
Olhando também no banco de dados, temos o mesmo resultado.



O mesmo ocorre para os demais endpoints:

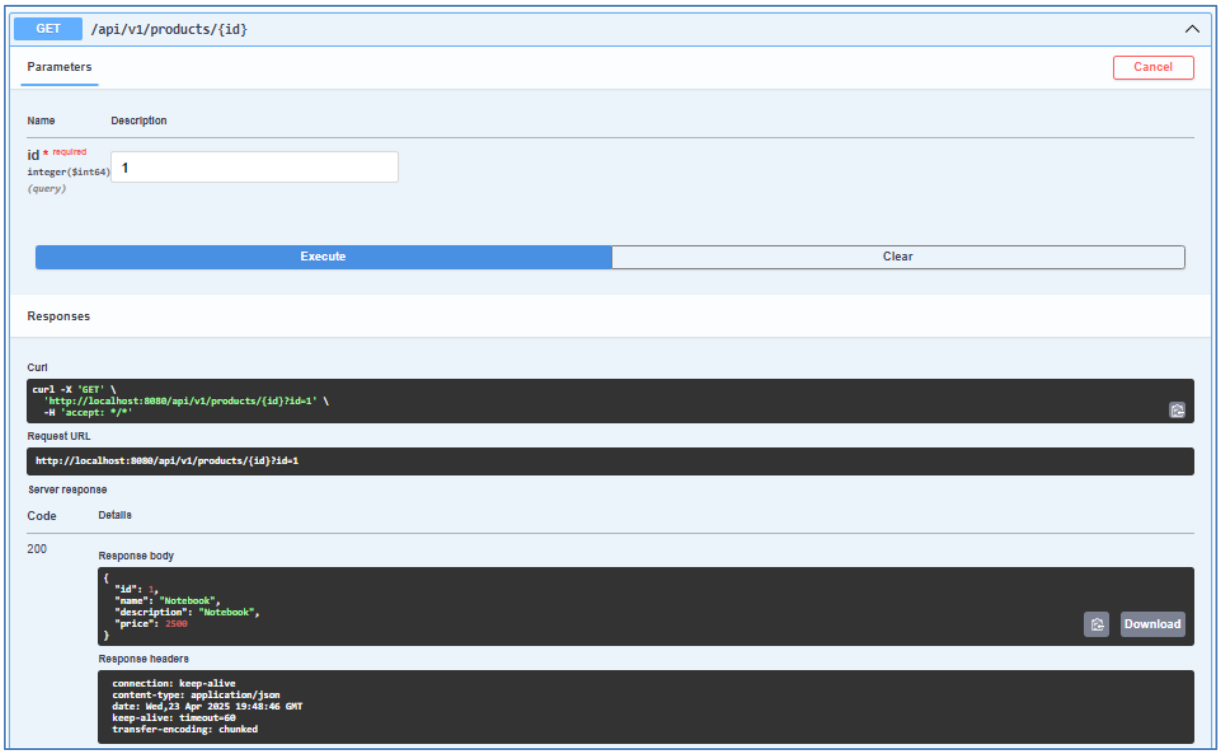
- **POST api/v1/products:**





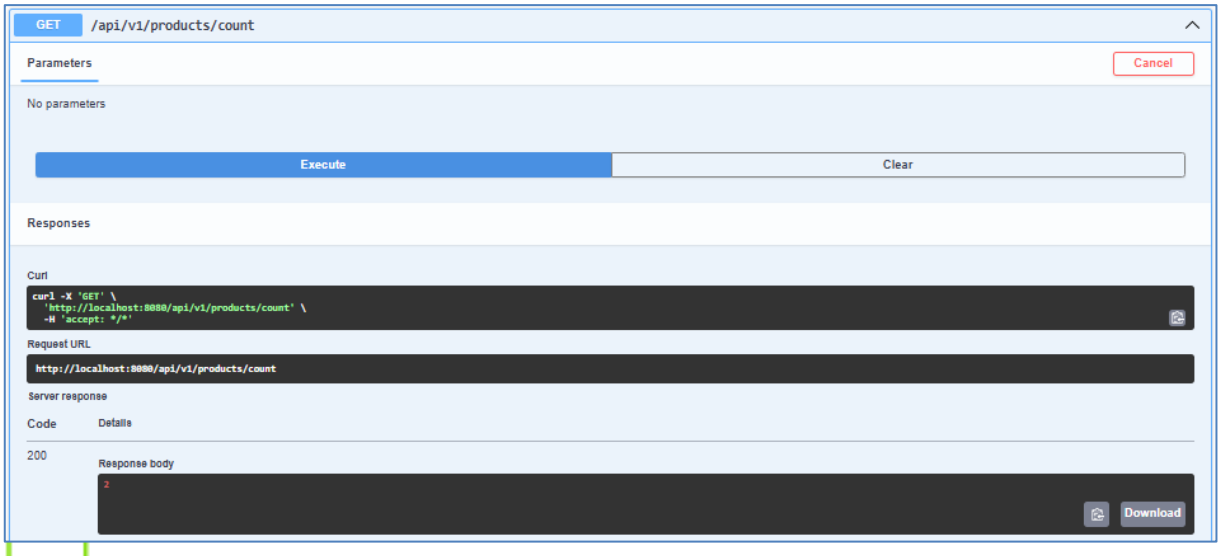
- **GET api/v1/products/{id}**

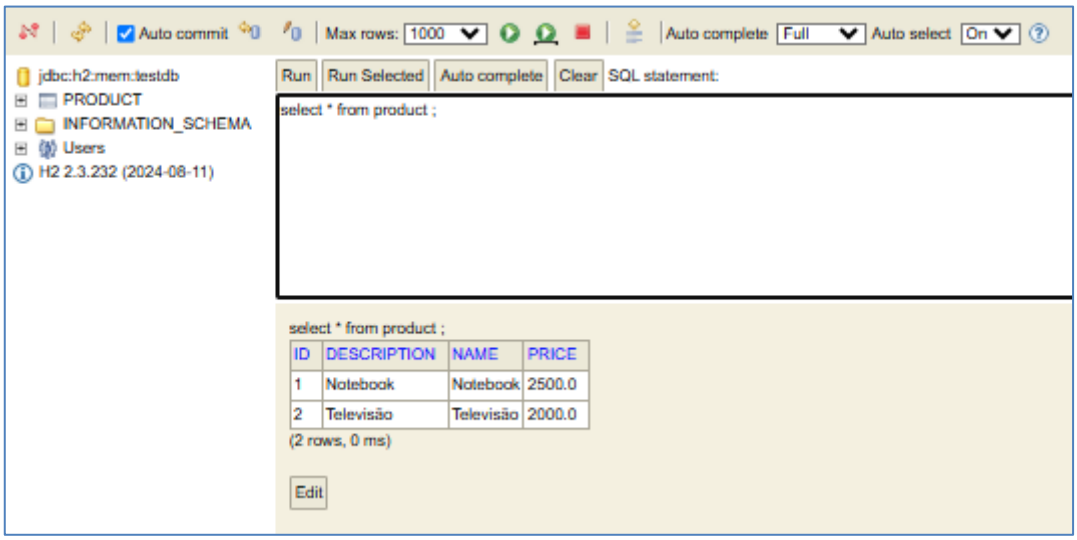
Nesse caso usando o id = 1, referente ao produto Notebook já cadastrado.



- **GET /api/v1/products/count**

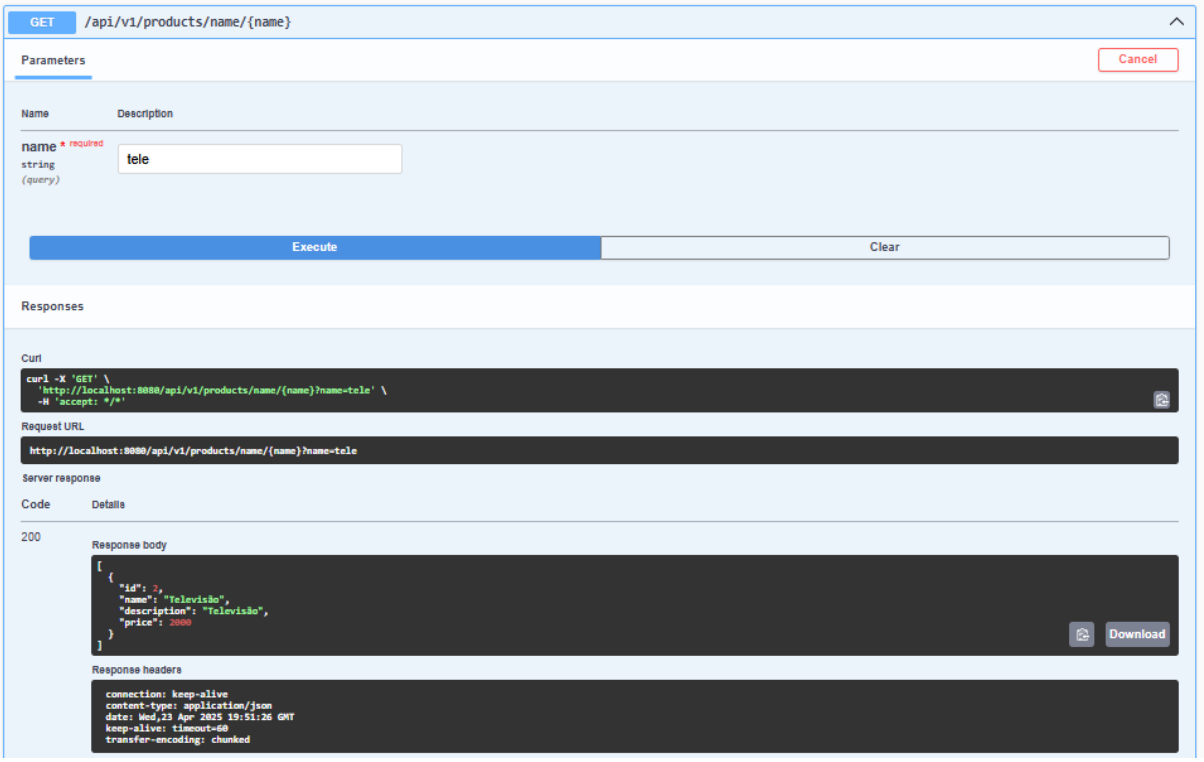
Neste caso está sendo retornado o valor 2, referente a quantidade de itens cadastrados no banco de dados.





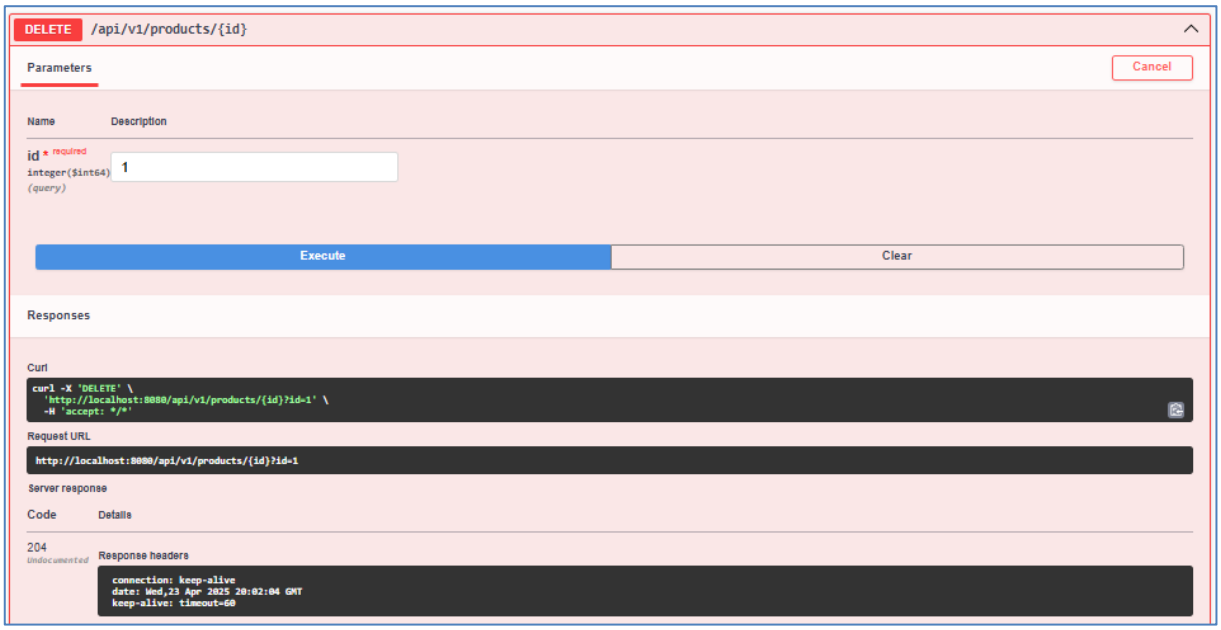
- **GET /api/v1/products/name/{name}**

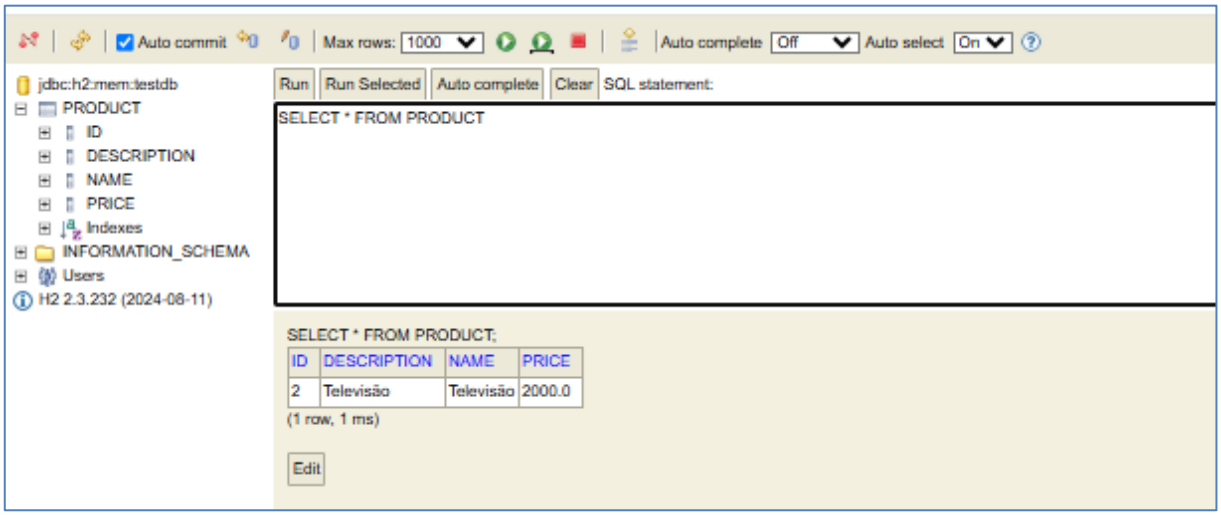
Nesse caso, foi usado a parte “tele” do nome do item “Televisão”, tendo em vista que a aplicação está configurada para identificar todos os itens que contem parte ou o todo do nome.



- **DELETE /api/v1/products/{id}**

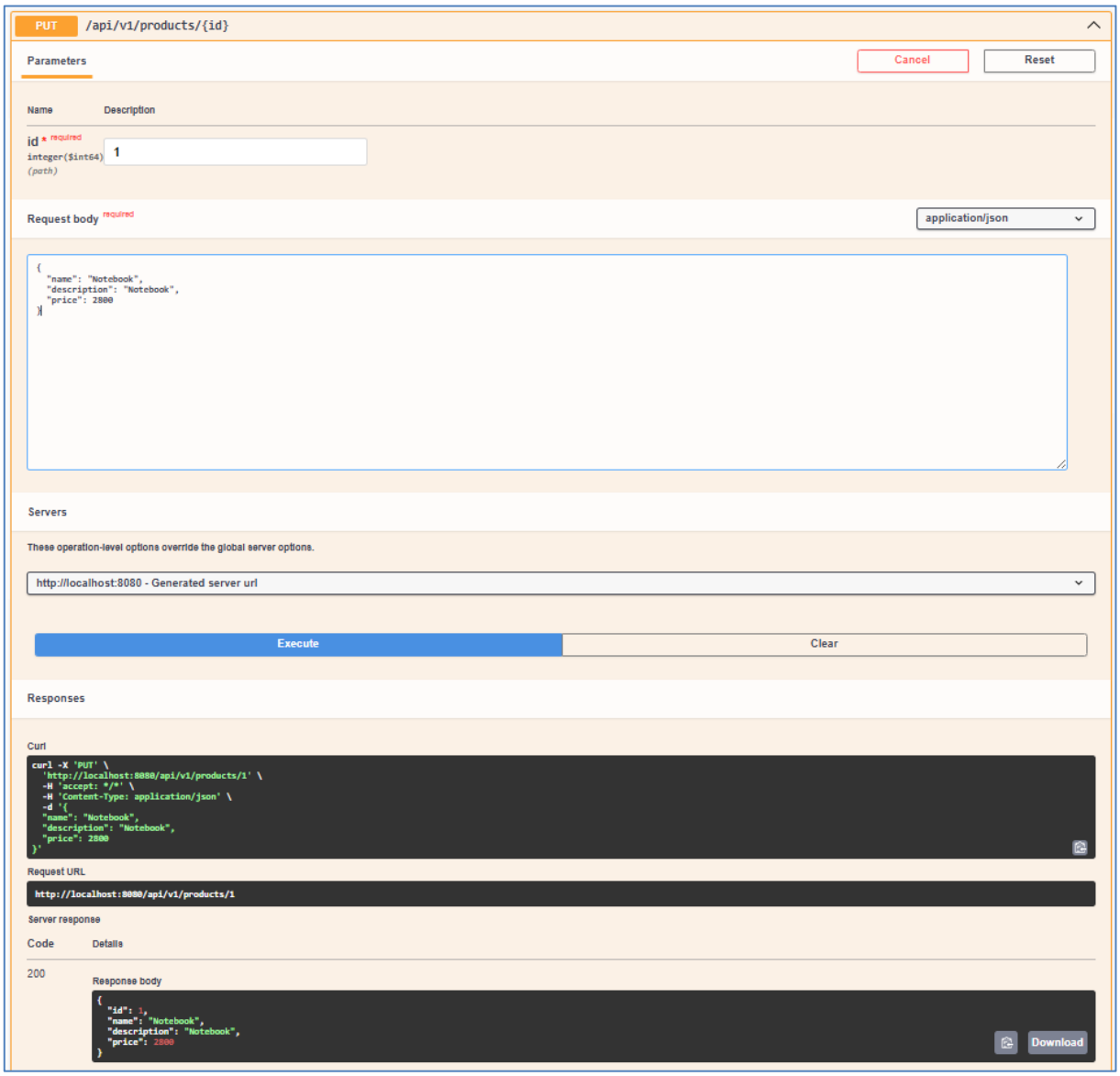
Nesse cenário, foi usado o id = 1, referente ao Notebook para ser removido.

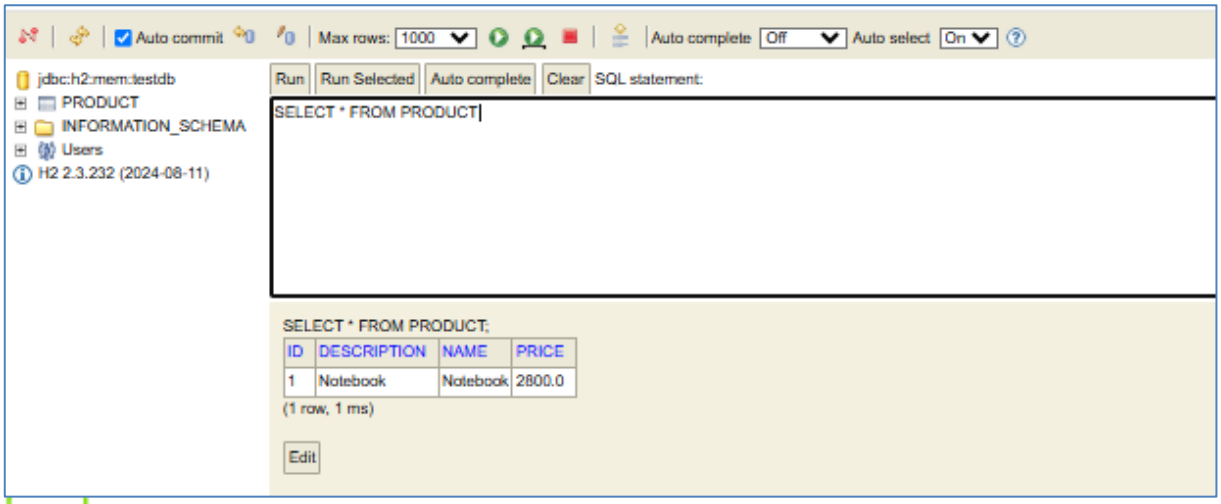




- **PUT /api/v1/products/{id}**

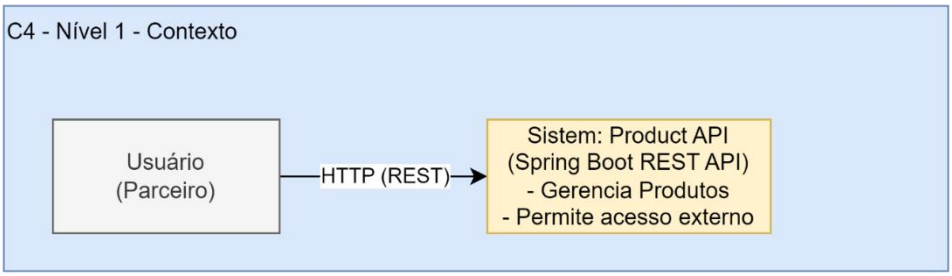
Nesse cenário, foi usado o id = 1, referente ao Notebook para ser atualizado.


PUT
/api/v1/products/{id}
Parameters
Cancel
Reset
Name
Description
id * required
integer($int64)
(path)
1
Request body required
application/json
{
"name": "Notebook",
"description": "Notebook",
"price": 2800
}
Servers
These operation-level options override the global server options.
http://localhost:8080 - Generated server url
Execute
Clear
Responses
Curl
curl -X 'PUT' \
'http://localhost:8080/api/v1/products/1' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"name": "Notebook",
"description": "Notebook",
"price": 2800
}'
Request URL
http://localhost:8080/api/v1/products/1
Server response
Code
Details
200
Response body
{
"id": 1,
"name": "Notebook",
"description": "Notebook",
"price": 2800
}
Download

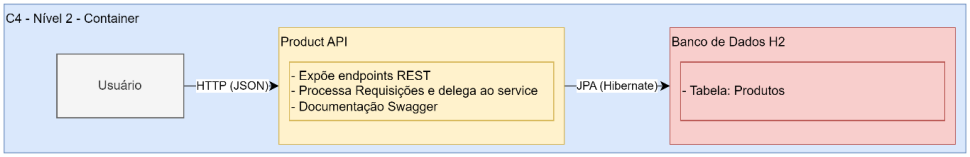

Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:testdb
PRODUCT
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUCT
SELECT * FROM PRODUCT;
ID
DESCRIPTION
NAME
PRICE
1
Notebook
Notebook
2800.0
(1 row, 1 ms)
Edit

10. Diagramas de Arquitetura (Modelo C4)

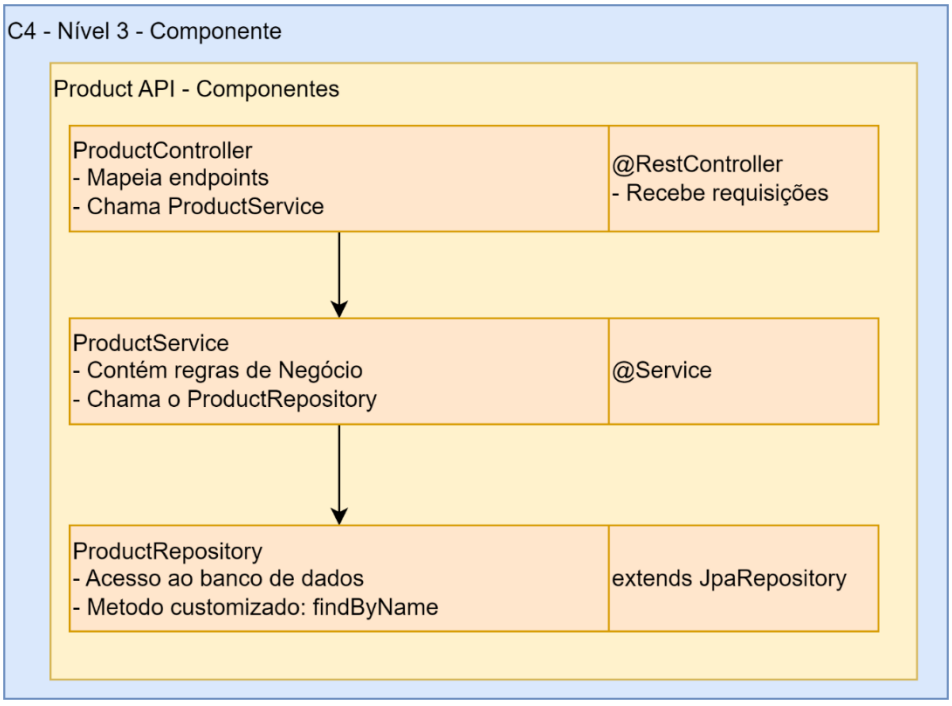
- **Nível 1 – Contexto**: Usuário acessa a API via HTTP.



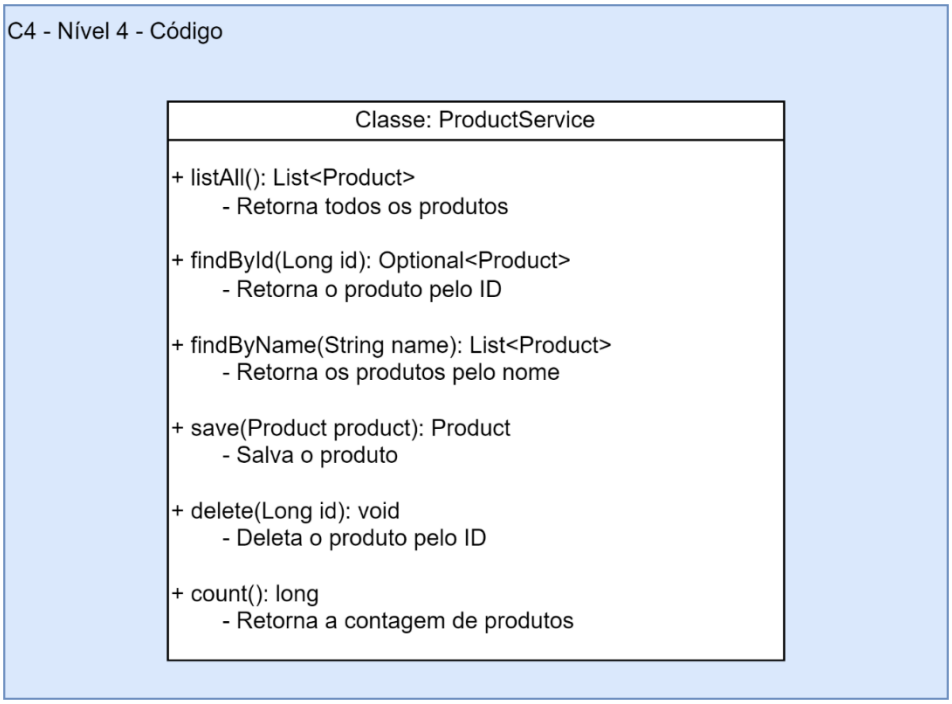
- **Nível 2 – Containers**: Aplicação Spring Boot + H2 Database.



- **Nível 3 – Componentes**: Controller, Service, Repository, Model.



- **Nível 4 – Código**: Classe ProdutoService com métodos de negócio.



Conclusão

Nesta seção, você deve resumir o seu desempenho ao longo do desafio final, destacando os principais pontos da sua experiência. A conclusão serve como uma oportunidade para refletir sobre o processo de resolução do desafio, abordando aspectos como o que foi aprendido, as dificuldades enfrentadas e as soluções implementadas.

Ao elaborar sua conclusão, considere os seguintes tópicos:

i. **Aplicação dos Conhecimentos:** explique como você utilizou os conhecimentos adquiridos durante o bootcamp para resolver os desafios propostos. Destaque as tecnologias, ferramentas e conceitos que foram fundamentais para o desenvolvimento da sua solução.

ii. **Principais Dificuldades e Superações:** relate os principais obstáculos que você encontrou durante a execução do desafio e como os superou. Isso demonstra sua capacidade de enfrentar problemas práticos e de adaptação diante de dificuldades.

iii. **Resultados Obtidos:** descreva os resultados gerais alcançados com a sua solução. Comente se todas as funcionalidades e requisitos foram implementados como esperado e se o resultado final atendeu aos objetivos propostos.

iv. **Lições Aprendidas:** reflita sobre os principais aprendizados que este desafio proporcionou, seja no aspecto técnico ou em habilidades, como gerenciamento de tempo, organização e trabalho em equipe (se aplicável).

v. **Melhorias Futuras:** se aplicável, mencione quais aspectos você considera que poderiam ser melhorados na sua solução ou no seu processo de trabalho, indicando possíveis ajustes que fariam a diferença em uma nova tentativa.

Essa seção deve ser escrita de forma clara e objetiva, demonstrando uma visão madura e crítica sobre o seu desempenho.

Conclusão por Paulo Gusttavo Tognato:

Esse foi um desafio bem completo e me serviu para colocar em prática diversos temas abordados durante o bootcamp. Gostei bastante do formato proposto, pois envolveu várias etapas reais para construção de um projeto prático, desde o início até a entrega final.

Durante o curso, tive a oportunidade de estudar diferentes modelos e estilos arquiteturais, além de me aprofundar nos princípios SOLID, padrões de projeto, práticas de DevOps, CI/CD, entre outros. Nesse projeto, consegui consolidar boa parte desses aprendizados utilizando o padrão arquitetural MVC com Java e Spring Boot. Também consegui aplicar conceitos dos princípios SOLID, como a separação de responsabilidades e a inversão de

dependência com o uso de interfaces e injeção de dependência. A estrutura utilizada no projeto foi baseada diretamente no que aprendi nos módulos.

Um dos pontos que considerei mais desafiadores foi a construção dos diagramas utilizando o modelo C4. Precisei pensar e representar a arquitetura da aplicação de uma forma clara e compreensível, o que me fez revisar bastante os conceitos. Outro desafio foi a configuração da documentação automatizada usando o Swagger com Spring. Tive alguns obstáculos no início, principalmente em relação à compatibilidade das dependências, mas com pesquisa e testes, consegui resolver e deixar funcionando como esperado.

O projeto ficou funcional, com uma estrutura bem organizada, código limpo e todos os requisitos implementados. A documentação gerada automaticamente pelo Swagger a partir dos métodos da classe de “controller” deixou o uso da aplicação bem mais claro, tanto para mim quanto para qualquer outro desenvolvedor que venha a utilizá-la. A separação em camadas (controller, service, repository) também ajudou bastante na organização geral.

Esse desafio reforçou a importância de organizar bem um projeto e documentar tudo da melhor forma possível. Também aprofundei meu entendimento sobre o padrão MVC e como cada camada tem sua responsabilidade bem definida.

Pensando em versões futuras, eu adicionaria testes automatizados, faria validações mais completas, trocaria o banco de dados em memória por um banco real, como MySQL, e colocaria a aplicação dentro de um contêiner com Docker. Além disso, seria interessante configurar um pipeline de CI/CD e fazer o deploy da aplicação em alguma plataforma de nuvem, como Heroku ou AWS.