Especificação Java

Importação do Projeto via Maven
Utilizando o Spring Tool Suite 4

<java.version>17</java.version>

Apache Tomcat®

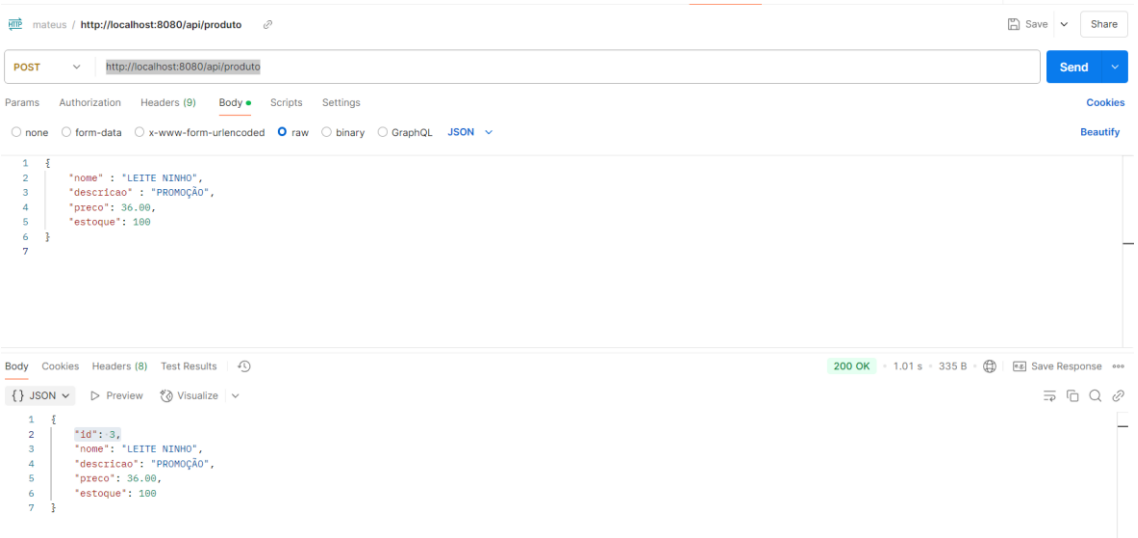
Atualizando dados na Base H2 temporário

Utilizado
Aplicacao Rest com Spring Boot: Módulo voltado para o desenvolvimento de aplicações com o padrão MVC, Utilize o padrão de projeto DTO, Utilize o padrão de projeto Factory para a criação dos DAOs, Escreva testes unitários para os serviços de criação, atualização, remoção e consulta de produtos, Utilize o padrão de projeto DAO (Data Access Object) para realizar operações no banco de dados, Utilize um banco de dados em memória (por exemplo, H2) para armazenar os dados dos produtos, Utilize o Spring Boot para criar a aplicação.

Requisitos Funcionais:

1. A aplicação deve permitir a criação de um novo produto com os seguintes dados: nome, descrição, preço e quantidade em estoque.
metodo: POST: http://localhost:8080/api/produto
Json:

```
{
    "nome" : "LEITE NINHO",
    "descricao" : "PROMOÇÃO",
    "preco": 36.00,
    "estoque": 100
}
```

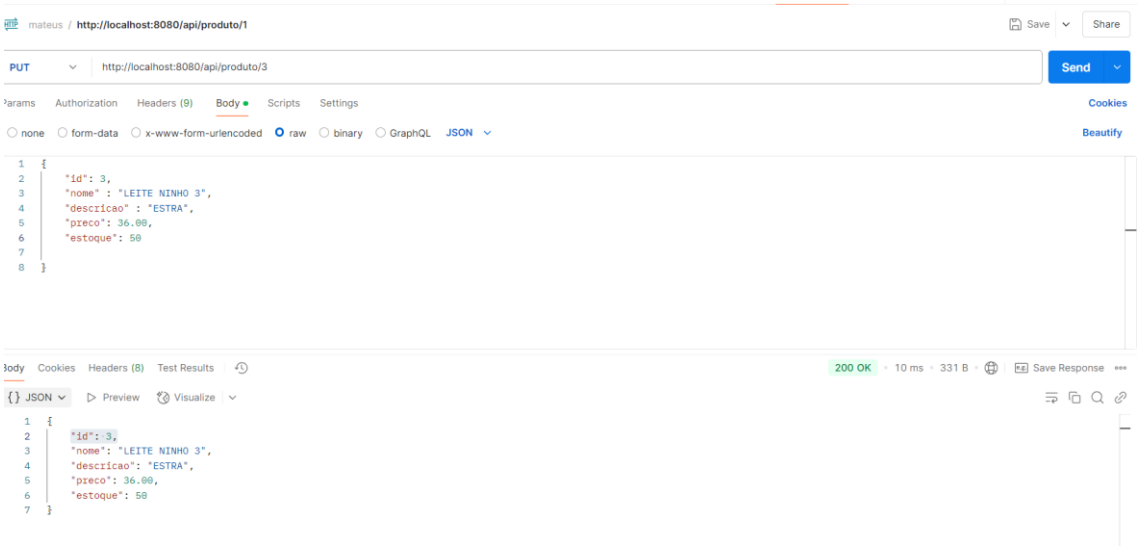


2. A aplicação deve permitir a atualização dos dados de um produto existente.

Metodo: PUT: http://localhost:8080/api/produto/3
Json

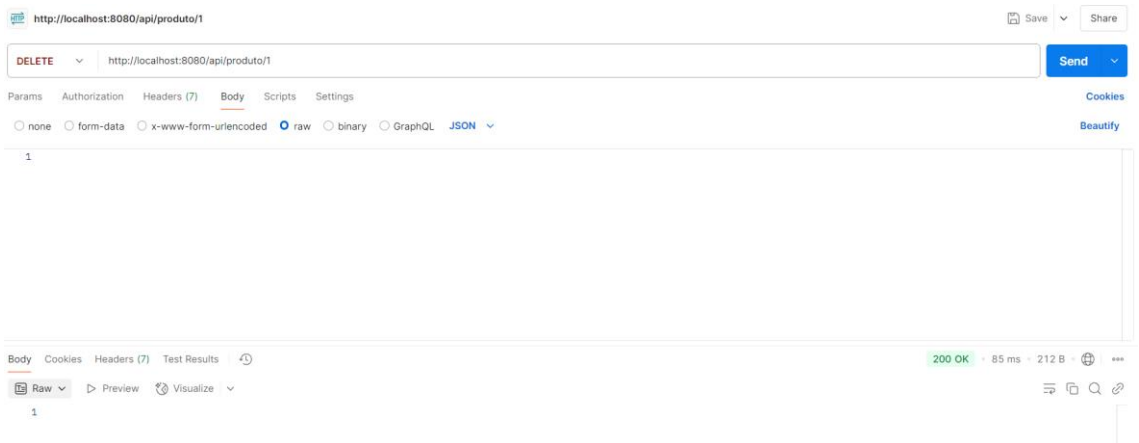
```
{
    "id": 3,
    "nome" : "LEITE NINHO 3",
    "descricao" : "EXTRA",
    "preco": 36.00,
    "estoque": 50

}
```



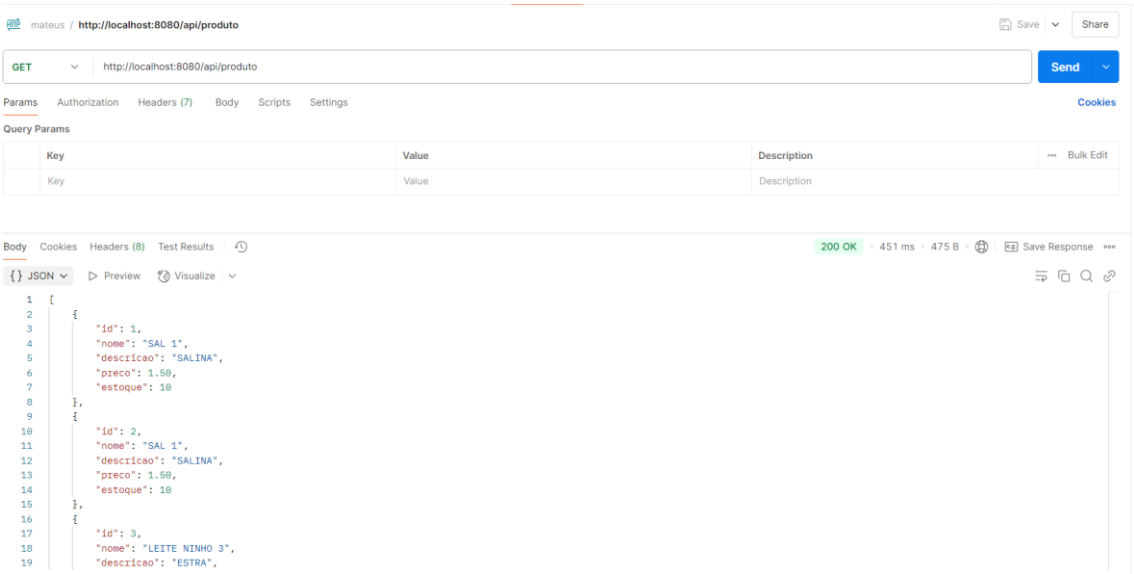
3. A aplicação deve permitir a remoção de um produto existente.

DELETE: http://localhost:8080/api/produto/3

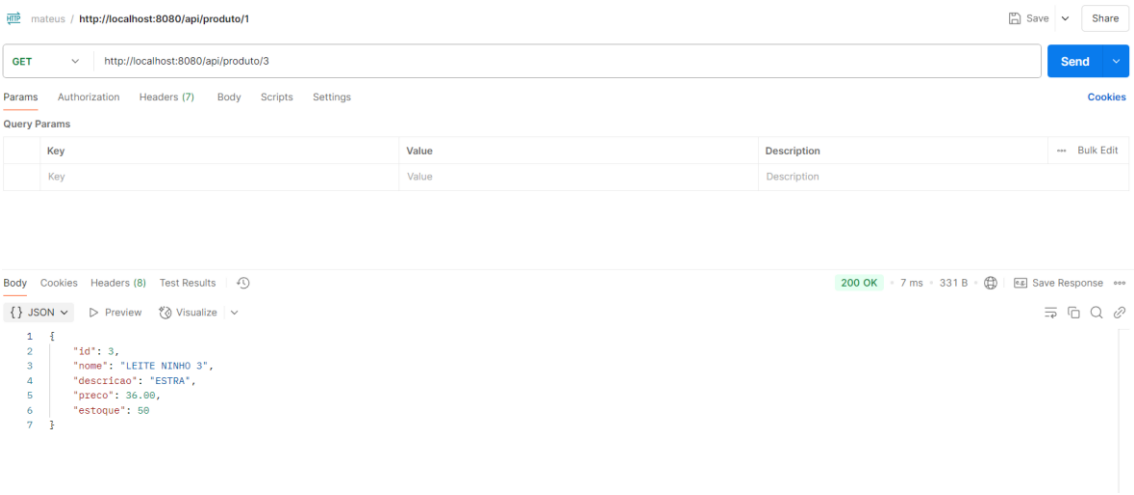


4. A aplicação deve permitir a consulta de todos os produtos cadastrados, bem como a consulta de um produto específico por ID.

GET: http://localhost:8080/api/produto



GET: http://localhost:8080/api/produto/3



5. Todos os endpoints devem seguir as boas práticas RESTful.

1. Estrutura dos Endpoints (Recursos e Nomes no Plural)

Os endpoints devem representar recursos, e os nomes devem ser substantivos no plural.

GET /produtos

GET /produtos/{id}

POST /produtos

PUT /produtos/{id}

DELETE /produtos/{id}

2. Métodos HTTP adequados

Use os métodos HTTP corretamente:

GET → Obter dados

POST → Criar um novo recurso

PUT → Atualizar um recurso **inteiro**

PATCH → Atualizar **parcialmente** um recurso

DELETE → Excluir um recurso

3.Uso correto dos códigos HTTP

Sempre retorne os códigos de status apropriados:

Código	Significado
200 OK	Requisição bem-sucedida (GET, PUT, PATCH)
201 Created	Recurso criado com sucesso (POST)
204 No Content	Recurso excluído com sucesso (DELETE)
400 Bad Request	Requisição inválida (exemplo: JSON malformado)
401 Unauthorized	Usuário não autenticado
403 Forbidden	Usuário sem permissão
404 Not Found	Recurso não encontrado
500 Internal Server Error	Erro no servidor

4.HATEOAS (Opcional, mas recomendado)

Para APIs mais robustas, você pode incluir links nos retornos:

```
{
  "id": 1,
  "nome": "Produto A",
  "preco": 100.0,
  "_links": {
    "self": { "href": "/produtos/1" },
    "delete": { "href": "/produtos/1" }
  }
}
```

5. Paginação e Filtros

Em listagens, evite retornar todos os dados. Use paginação:

http

Copiar código

GET /produtos?page=1&size=10&sort=nome,asc

6. Versionamento da API

Se sua API precisar de versionamento, siga um desses métodos:

1. Via URL

Copiar código

GET /v1/produtos

GET /v2/produtos

2. Via Header

http

Copiar código

GET /produtos

Headers: Accept: application/vnd.meuapp.v1+json

Testes unitários

Testes unitários cobrem os serviços com Mockito.

Testes de integração validam a API REST.

Cobertura de código pode ser analisada com JaCoCo.

mvn clean test

mvn jacoco:report

Carlos Antônio de Almeida Filho
Analista de sistemas