

Desafio Final - Caio Reimberg

A aplicação escolhida foi uma API REST para manipulação e consulta de Pedidos utilizando SpringBoot Java e um repositório H2 como base de dados para persistência.

Foi utilizado o site Spring Initializr para gerar um pacote Maven que posteriormente foi importado em uma IDE (Eclipse IDE) a fim de gerar a aplicação Java com repositório, e permitir validar as APIs geradas e resultados esperados.

Abaixo podemos ver os diagramas C4 da aplicação

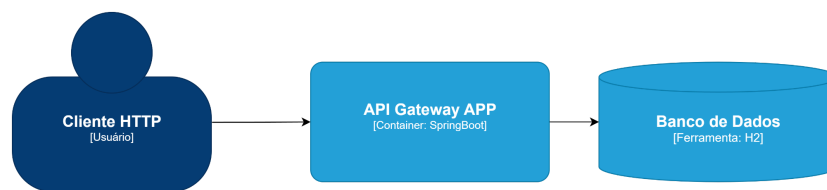


Imagem 1: Diagrama de Contexto

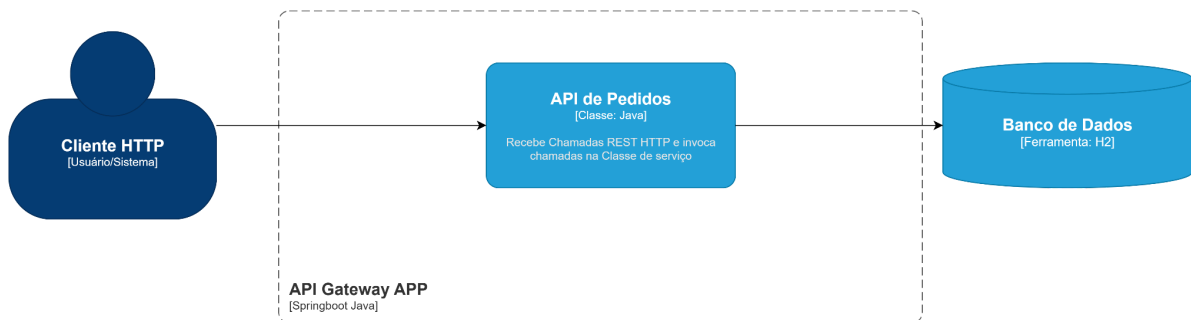


Imagem 2: Diagrama de Contêineres

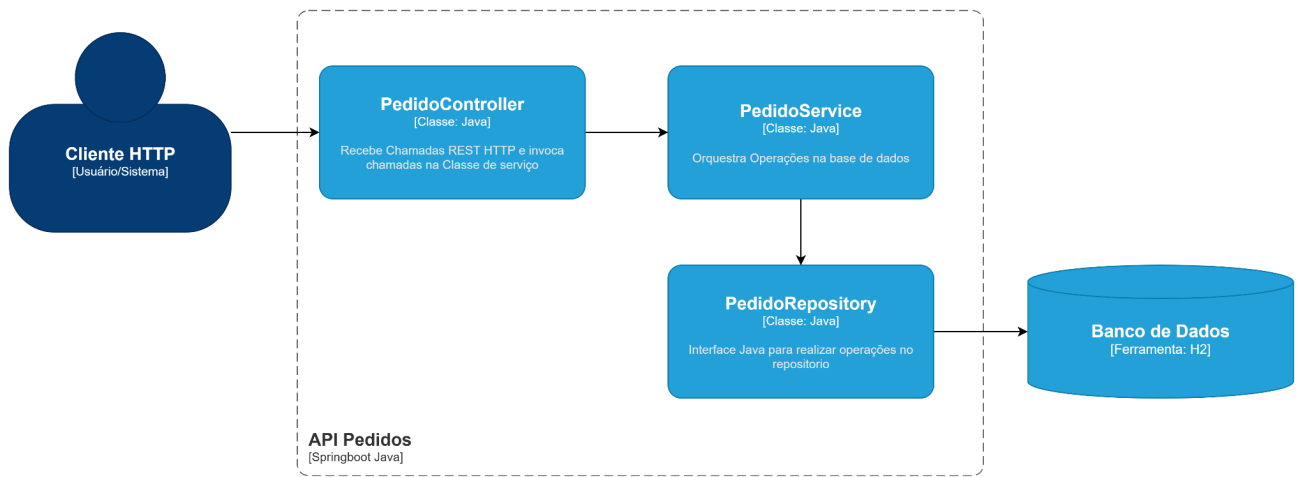


Imagem 3: Diagrama de Componentes

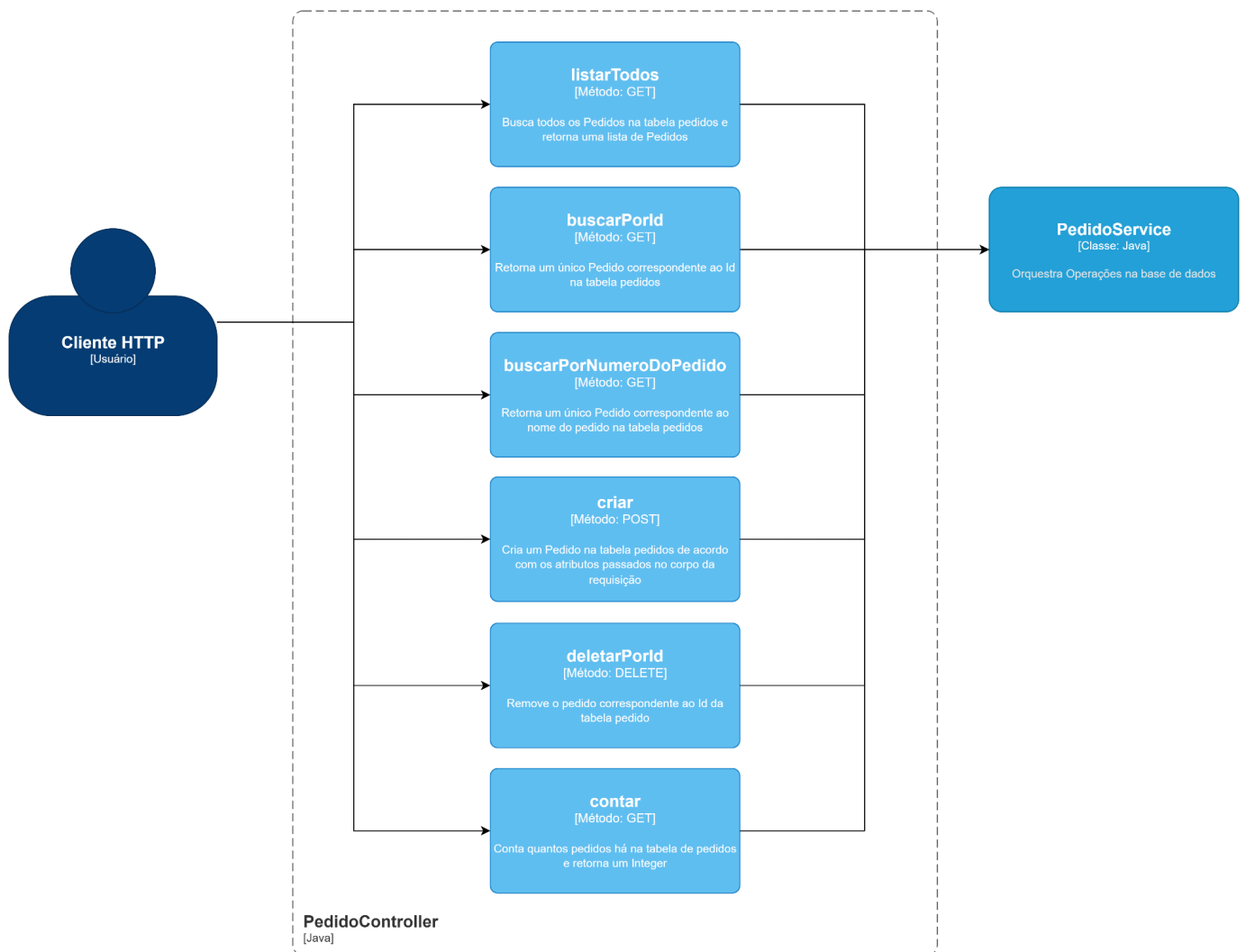


Imagem 4: Diagrama de Código - PedidoController

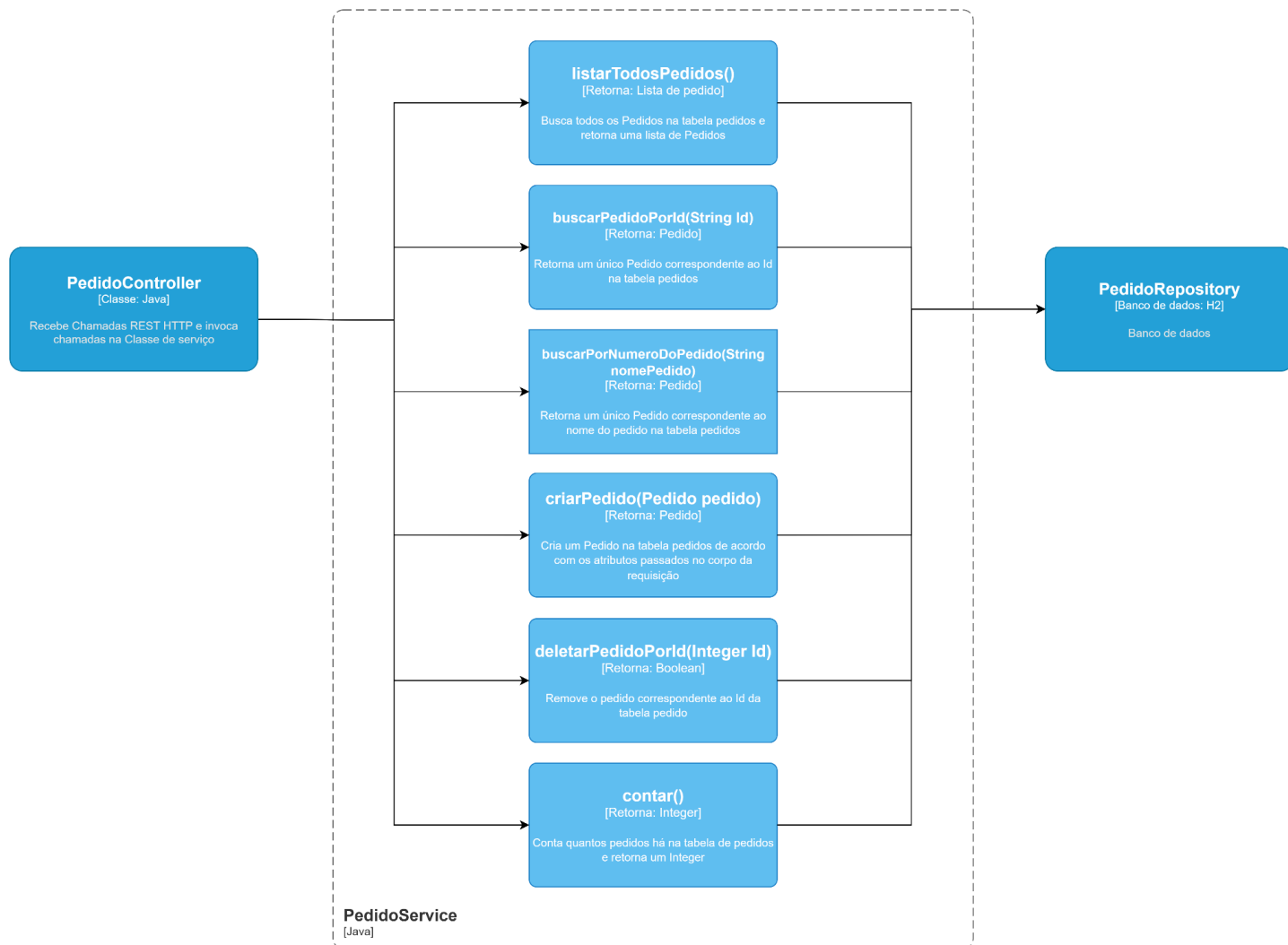
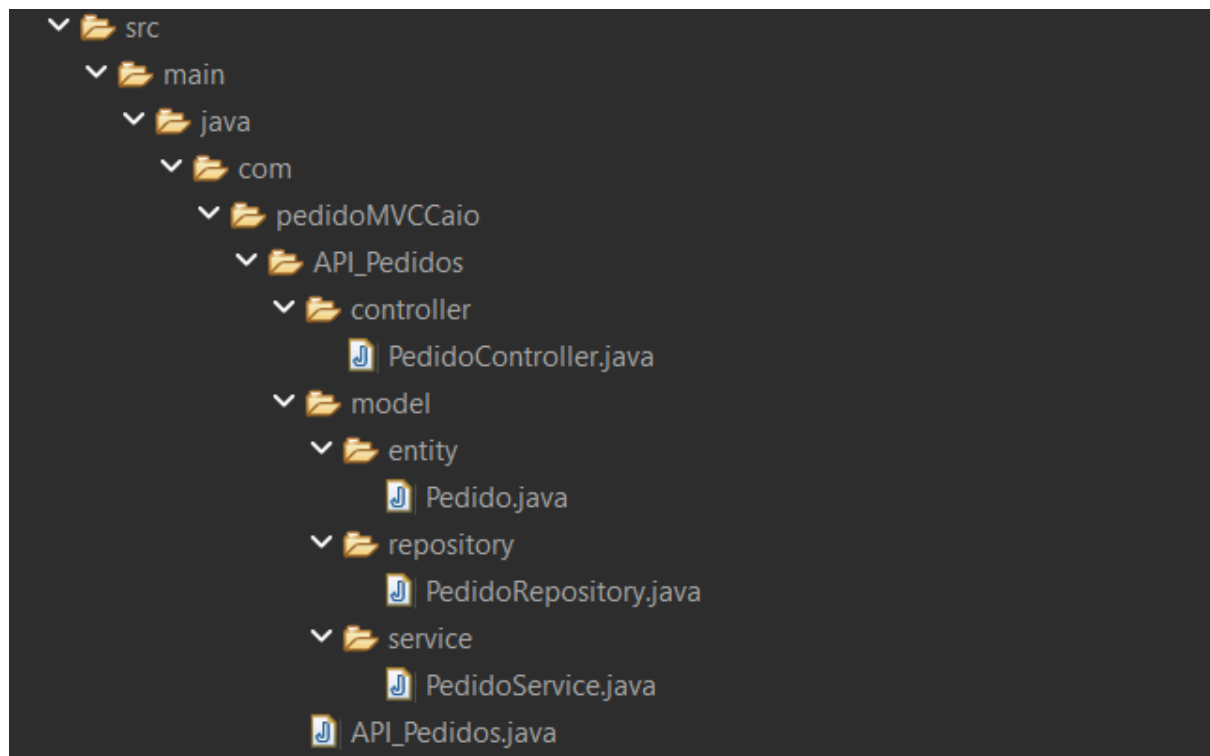


Imagem 4: Diagrama de Código - PedidoService

A estrutura do projeto foi organizado seguindo as diretrizes do MVC clássico, separando as responsabilidades entre a camada de Model e Controller (a camada de view não foi utilizada nesse caso já que se trata apenas de uma API REST).



Na tabela abaixo podemos ver as responsabilidades que buscamos alcançar com cada camada do repositório:

Camada	Pasta	Responsabilidade
Controller	Controller	Camada que descreve os métodos HTTP REST recebidos, traduz os parâmetros, retorna uma resposta de acordo com a operação realizada
Model	Entity	Aqui estão as entidades de domínio, descrevendo os campos pertencentes a cada uma.
Model	Service	Essa camada concentra a lógica de negócio e orquestra as operações a serem realizadas no repositório.
Model	Repository	Abstrai acesso a dados e permite trabalhar com persistência de dados (Spring Data JPA, JDBC por exemplo).

Tabela 1: Descrição das pastas e responsabilidades

Após montar as classes e os métodos sugeridos no enunciado, foi possível testar as chamadas utilizando o Postman e validar a interação entre as chamadas realizadas e o banco de dados.

1. Listar Todos os Pedidos

Endpoint: /v1/pedidos/

Método: GET

Resposta: List<Pedido>

The screenshot shows a Postman interface for a GET request to `http://localhost:8080/v1/pedidos/`. The response is a 200 OK status with a 220 ms latency and 498 B of data. The response body is a JSON array containing three order objects.

Key	Value	Description
Key	Value	Description

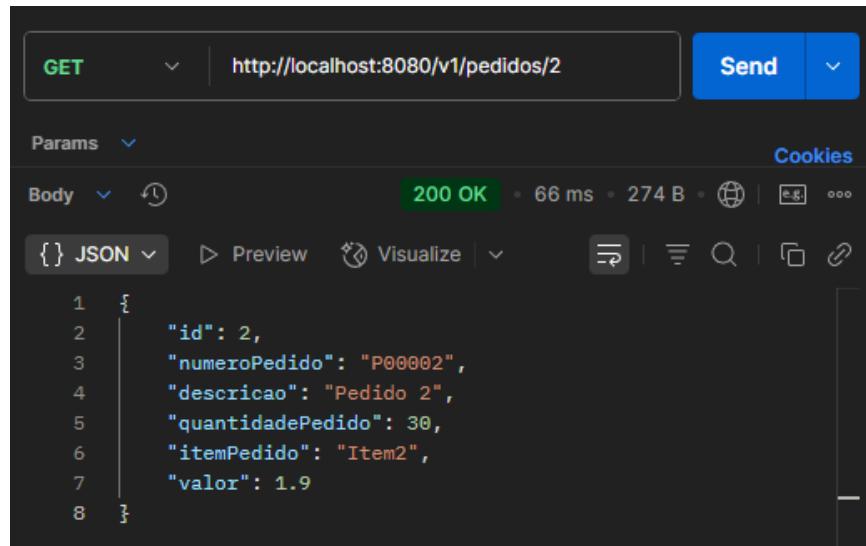
```
1  [
2    {
3      "id": 1,
4      "numeroPedido": "P000001",
5      "descricao": "Pedido 1",
6      "quantidadePedido": 1,
7      "itemPedido": "Item1",
8      "valor": 9.9
9    },
10   {
11     "id": 2,
12     "numeroPedido": "P000002",
13     "descricao": "Pedido 2",
14     "quantidadePedido": 30,
15     "itemPedido": "Item2",
16     "valor": 1.9
17   },
18   {
19     "id": 3,
20     "numeroPedido": "P000003",
21     "descricao": "Pedido 3",
22     "quantidadePedido": 12,
23     "itemPedido": "Item3",
24     "valor": 10.0
25   }
26 ]
```

2. Buscar Por Id

Endpoint: /v1/pedidos/{id}

Método: GET

Resposta: Pedido

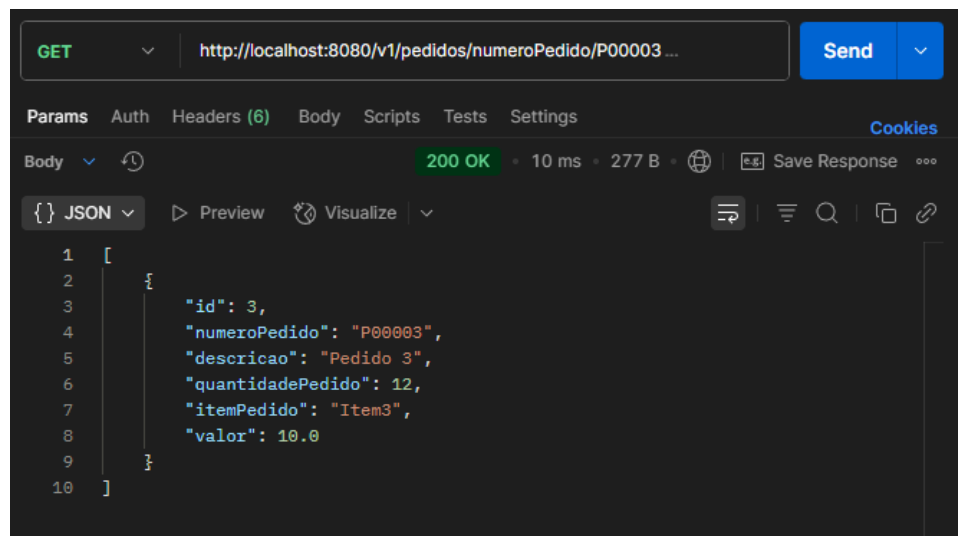


3. Buscar Por Numero Do Pedido

Endpoint: /v1/pedidos/numeroPedido/{numeroPedido}

Método: GET

Resposta: List<Pedido>

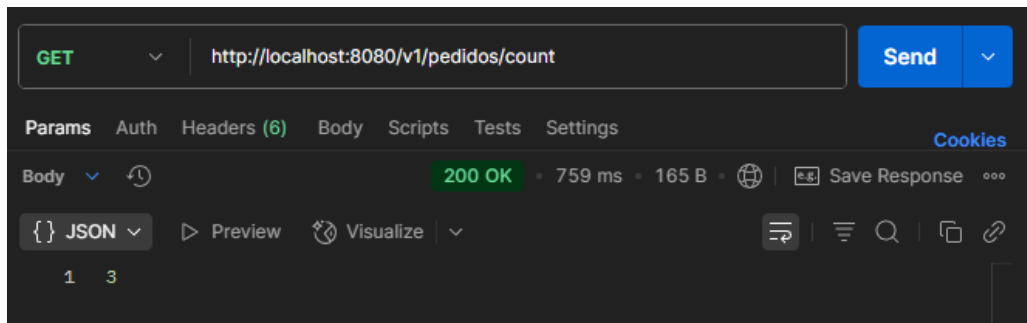


4. Contagem de pedidos

Endpoint: /v1/pedidos/numeroPedido/{numeroPedido}

Método: GET

Resposta: Integer

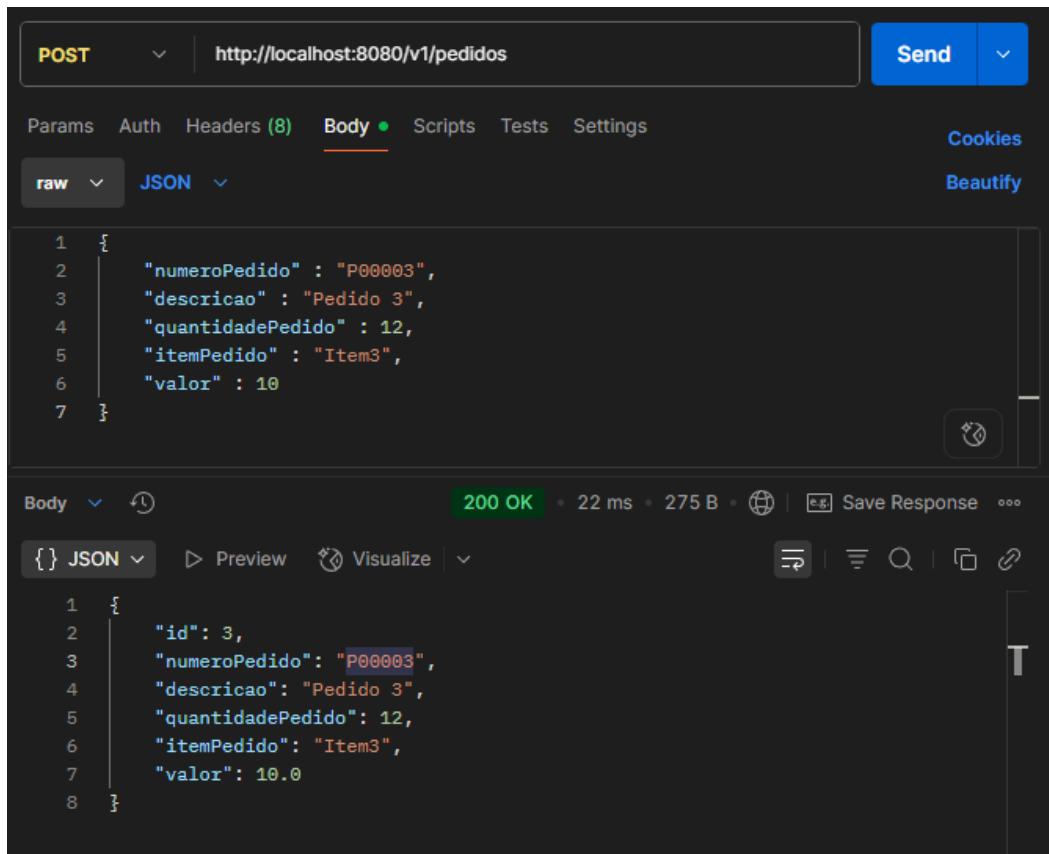


5. Criar pedido

Endpoint: /v1/pedidos

Método: POST

Resposta: Integer



6. Deletar Pedido

Endpoint: /v1/pedidos/{id}

Método: DELETE

Resposta: Boolean

