Spring Boot

Criando a camada DTO (Data Transfer Object)

PROF. ME. JEFFERSON PASSERINI

- O padrão DTO (Data Transfer Object) é um padrão de design utilizado no desenvolvimento de software para transferir dados entre diferentes partes de um sistema ou entre sistemas distintos.

- Este padrão é particularmente útil quando se deseja reduzir o número de chamadas de rede ou operações de I/O, uma vez que permite agrupar múltiplos valores em um único objeto.

- Principais características:

1. **Simples e Leve**: Os DTOs são objetos simples que não contêm lógica de negócios ou comportamento além de getters e setters para encapsular os dados. Eles geralmente possuem apenas atributos públicos ou propriedades com métodos de acesso.
2. **Serialização**: DTOs são frequentemente usados para serialização e desserialização de dados, especialmente em aplicações que utilizam APIs RESTful ou serviços web, onde os dados precisam ser convertidos para formatos como JSON ou XML.
3. **Isolamento**: Ao utilizar DTOs, a estrutura interna do sistema pode ser isolada dos detalhes da comunicação externa, permitindo mudanças internas sem impactar diretamente os consumidores externos dos dados.
4. **Eficiência**: DTOs ajudam a melhorar a eficiência de comunicação, pois permitem a transferência de dados em lotes, reduzindo a necessidade de múltiplas chamadas de rede.

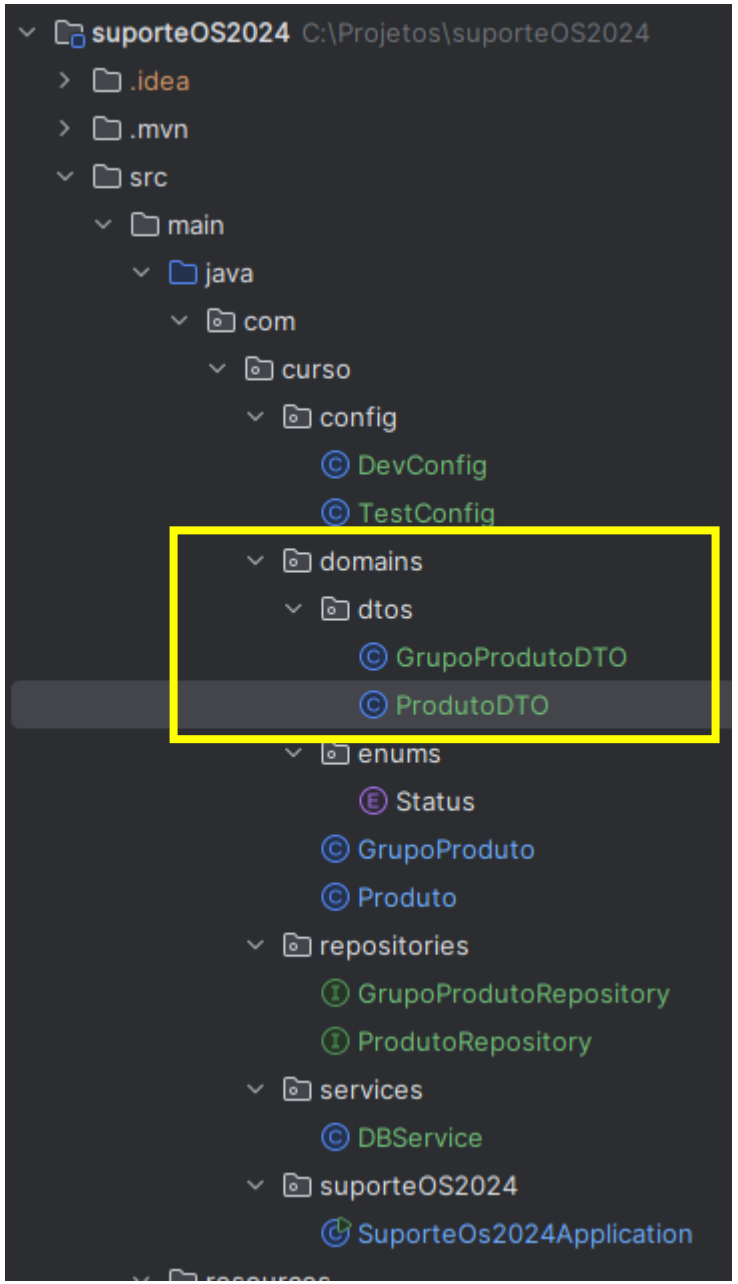
- Exemplo:
- Imagine um sistema de gerenciamento de pedidos onde um cliente deseja obter informações sobre um pedido específico.
- Em vez de fazer múltiplas chamadas para obter dados de diferentes entidades relacionadas ao pedido (como detalhes do cliente, produtos, pagamentos, etc.), um único DTO pode ser utilizado para transferir todos esses dados de uma vez.
- Neste exemplo PedidoDTO, agrega informações de um pedido, incluindo uma lista de ProdutoDTO que contém detalhes sobre os produtos do pedido.
- Este DTO pode ser enviado de uma camada do servidor para uma camada de apresentação, ou se um serviço de backend para um front end.

```java
public class PedidoDTO {
    private Long idPedido;
    private String nomeCliente;
    private List<ProdutoDTO> produtos;
    private Double valorTotal;

    // Getters e Setters
}

public class ProdutoDTO {
    private Long idProduto;
    private String nomeProduto;
    private Double preco;

    // Getters e Setters
}
```

- Benefícios:

1. **Facilita a Manutenção**: DTOs ajudam a separar as preocupações, tornando o código mais modular e fácil de manter.
2. **Melhora a Performance**: Ao reduzir o número de chamadas de rede, a performance do sistema é otimizada.
3. **Flexibilidade**: Permite modificar a estrutura interna sem afetar as interfaces externas, oferecendo maior flexibilidade na evolução do sistema.

- Em resumo, o padrão DTO é uma prática recomendada para transferir dados entre diferentes partes de um sistema ou entre sistemas distintos, promovendo eficiência, clareza e manutenção facilitada.



- Em nosso projeto crie um subpacote denominado **dtos** dentro do pacote **domains**.

- Dentro de **dtos** crie a classe java **GrupoProdutoDTO e ProdutoDTO.**

```java
package com.curso.domains.dtos;

import com.curso.domains.GrupoProduto;
import com.curso.domains.enums.Status;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.JoinColumn;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

public class GrupoProdutoDTO {  no usages  new *

    private int id;  3 usages

    @NotNull(message = "O campo descrição não pode ser nulo")  3 usages
    @NotBlank(message = "O campo descrição não pode estar vazio")
    private String descricao;
    private int status;  3 usages

    public GrupoProdutoDTO(){}  no usages  new *

    public GrupoProdutoDTO(GrupoProduto grupoProduto){  no usages  new *
        this.id = grupoProduto.getId();
        this.descricao = grupoProduto.getDescricao();
        this.status = grupoProduto.getStatus().getId();
    }

    public int getId() {  no usages  new *
        return id;
    }
```

- Implemente a classe **GrupoProdutoDTO**.

- Observe que o construtor da classe converte do domain de GrupoProduto para GrupoProdutoDTO.

- Getter e Setter de GrupoProduto DTO

```
31
32      public void setId(int id) {  no usages  new *
33          this.id = id;
34      }
35
36      public @NotNull(message = "O campo descricão não pode ser nulo") @NotBlank(message = "O campo descrição não pode estar vazio") String getDescricao() {  no usages  new *
37          return descricao;
38      }
39
40      public void setDescricao(@NotNull(message = "O campo descricão não pode ser nulo") @NotBlank(message = "O campo descrição não pode estar vazio") String descricao) {
41          this.descricao = descricao;
42      }
43
44      public int getStatus() {  no usages  new *
45          return status;
46      }
47
48      public void setStatus(int status) {  no usages  new *
49          this.status = status;
50      }
51  }
```

```java
public class ProdutoDTO {  no usages  new *

    private long idProduto;  3 usages

    @NotNull(message = "O campo descricão não pode ser nulo")  3 usages
    @NotBlank(message = "O campo descrição não pode estar vazio")
    private String descricao;

    @NotNull(message = "O campo saldoEstoque não pode ser nulo")  3 usages
    @Digits(integer = 15, fraction = 3)
    private BigDecimal saldoEstoque;

    @NotNull(message = "O campo valorUnitario não pode ser nulo")  3 usages
    @Digits(integer = 15, fraction = 3)
    private BigDecimal valorUnitario;

    @NotNull(message = "O campo valorEstoque não pode ser nulo")  3 usages
    @Digits(integer = 15, fraction = 2)
    private BigDecimal valorEstoque;

    @JsonFormat(pattern = "dd/MM/yyyy")  3 usages
    private LocalDate dataCadastro = LocalDate.now();

    @NotNull(message = "O campo Grupo Produto é requerido")  3 usages
    private int grupoProduto;
    private String descricaoGrupoProduto;  3 usages

    private int status;  3 usages
```

- Implemente a classe **ProdutoDTO**.

- Observe que o construtor da classe converte do domain de Produto para ProdutoDTO.

```
43
44      public ProdutoDTO(){ }  no usages  new *
45
46 @    public ProdutoDTO(Produto produto){  no usages  new *
47          this.idProduto = produto.getIdProduto();
48          this.descricao = produto.getDescricao();
49          this.valorUnitario = produto.getValorUnitario();
50          this.saldoEstoque = produto.getSaldoEstoque();
51          this.valorEstoque = produto.getValorEstoque();
52          this.dataCadastro = produto.getDataCadastro();
53          this.grupoProduto = produto.getGrupoProduto().getId();
54          this.descricaoGrupoProduto = produto.getGrupoProduto().getDescricao();
55          this.status = produto.getStatus().getId();
56      }
57
```

- Implemente a classe **ProdutoDTO**.

- Observe que o construtor da classe converte do domain de Produto para ProdutoDTO.

```java
    public long getIdProduto() {  no usages  new *
        return idProduto;
    }

    public void setIdProduto(long idProduto) {  no usages  new *
        this.idProduto = idProduto;
    }

    public @NotNull(message = "O campo descricão não pode ser nulo") @NotBlank(message = "O campo descrição não pode estar vazio") String getDescricao() {  no usages  new *
        return descricao;
    }

    public void setDescricao(@NotNull(message = "O campo descricão não pode ser nulo") @NotBlank(message = "O campo descrição não pode estar vazio") String descricao) {
        this.descricao = descricao;
    }

    public @NotNull(message = "O campo saldoEstoque não pode ser nulo") @Digits(integer = 15, fraction = 3) BigDecimal getSaldoEstoque() {  no usages  new *
        return saldoEstoque;
    }

    public void setSaldoEstoque(@NotNull(message = "O campo saldoEstoque não pode ser nulo") @Digits(integer = 15, fraction = 3) BigDecimal saldoEstoque) {  no usages  new
        this.saldoEstoque = saldoEstoque;
    }

    public @NotNull(message = "O campo valorUnitario não pode ser nulo") @Digits(integer = 15, fraction = 3) BigDecimal getValorUnitario() {  no usages  new *
        return valorUnitario;
    }

    public void setValorUnitario(@NotNull(message = "O campo valorUnitario não pode ser nulo") @Digits(integer = 15, fraction = 3) BigDecimal valorUnitario) {  no usages
        this.valorUnitario = valorUnitario;
    }

    public @NotNull(message = "O campo valorEstoque não pode ser nulo") @Digits(integer = 15, fraction = 2) BigDecimal getValorEstoque() {  no usages  new *
        return valorEstoque;
    }

    public void setValorEstoque(@NotNull(message = "O campo valorEstoque não pode ser nulo") @Digits(integer = 15, fraction = 2) BigDecimal valorEstoque) {  no usages  new
        this.valorEstoque = valorEstoque;
    }
```

Implemente a classe **ProdutoDTO**.

Observe que o construtor da classe converte do domain de Produto para ProdutoDTO.

```
97
98      public LocalDate getDataCadastro() {  no usages  new *
99          return dataCadastro;
100     }
101
102     public void setDataCadastro(LocalDate dataCadastro) {  no usages  new *
103         this.dataCadastro = dataCadastro;
104     }
105
106     @NotNull(message = "O campo Grupo Produto é requerido")  no usages  new *
107     public int getGrupoProduto() {
108         return grupoProduto;
109     }
110
111     public void setGrupoProduto(@NotNull(message = "O campo Grupo Produto é requerido") int grupoProduto) {
112         this.grupoProduto = grupoProduto;
113     }
114
115     public String getDescricaoGrupoProduto() {  no usages  new *
116         return descricaoGrupoProduto;
117     }
118
119     public void setDescricaoGrupoProduto(String descricaoGrupoProduto) {  no usages  new *
120         this.descricaoGrupoProduto = descricaoGrupoProduto;
121     }
122
123     public int getStatus() {  no usages  new *
124         return status;
125     }
126
127     public void setStatus(int status) {  no usages  new *
128         this.status = status;
129     }
130 }
```

Criando Projeto Spring Boot – DTO

- Implemente a classe **ProdutoDTO**.

- Observe que o construtor da classe converte do domain de Produto para ProdutoDTO.

Spring Boot

Criando a camada Services – findAll( )

PROF. ME. JEFFERSON PASSERINI

- A função principal da camada Service é orquestrar a lógica de negócios da aplicação e garantir que as regras de negócio sejam aplicadas corretamente.
- Funcões:

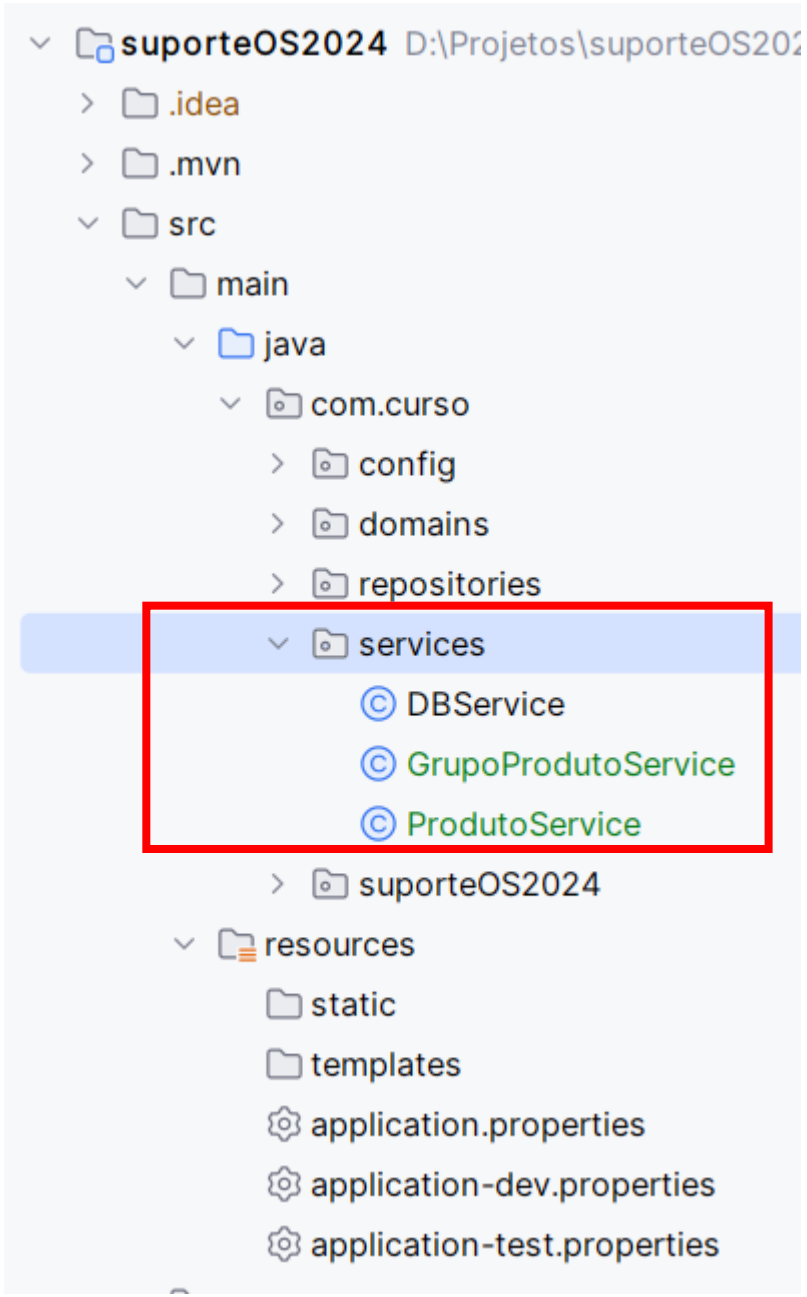
1. **Intermediar Requisições**: as classes de serviço recebem as chamadas dos recursos (Resources ou Controllers) e intermediam essas requisições, encaminhando-as para os repositórios (Repositories) ou outras camadas necessárias.
2. **Aplicar a Lógica de Negócio**: responsável por implementar a lógica de negócios da aplicação. Isso inclui validações, cálculos, manipulação de dados e quaisquer outras regras específicas que governem como os dados devem ser tratados e processados.
3. **Gerenciar Transações**: em muitas aplicações, a camada Service gerencia transações, garantindo que operações que precisam ser atômicas sejam tratadas corretamente, utilizando mecanismos de commit e rollback conforme necessário.

- Funcões (Continuação):

4. **Comunicação entre Repositórios**: os serviços podem interagir com múltiplos repositórios para compor dados de diferentes fontes. Por exemplo ao criar um pedido, um serviço pode precisar acessar repositórios de clientes, produtos e pedidos.

5. **Encapsular Complexidade**: ao encapsular a complexidade da lógica de negócios dentro dos serviços, o código nos recursos e repositórios se torna mais simples e focado nas responsabilidades específicas.

6. **Centralizar a Regra de Negócio**: ter as regras de negócio centralizadas na camada Service facilita a manutenção e a evolução do sistema, uma vez que as mudanças nas regras de negócio são feitas em um único lugar.



- No pacote **Services** crie as classes java denominadas **GrupoProdutoService e ProdutoService.**

```java
package com.curso.services;

import com.curso.domains.dtos.GrupoProdutoDTO;
import com.curso.repositories.GrupoProdutoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service  no usages  new *
public class GrupoProdutoService {

    @Autowired  1 usage
    private GrupoProdutoRepository grupoProdutoRepo;

    public List<GrupoProdutoDTO> findAll(){  no usages  new *
        //retorna uma lista de ProdutoDTO
        return grupoProdutoRepo.findAll().stream() Stream<Grupo
                .map(obj -> new GrupoProdutoDTO(obj)) Stream<Gr
                .collect(Collectors.toList());
    }
}
```

- Implemente a **GrupoProdutoService.**
  - **A camada Service** solicita a repository uma lista de GrupoProduto, converte para GrupoProdutoDTO e retorna a lista de DTOs para a resource (camada que estarão nossos endpoints – api´s) que a solicitou.

```java
package com.curso.services;

import java.util.List;
import java.util.stream.Collectors;

import com.curso.domains.dtos.ProdutoDTO;
import com.curso.repositories.ProdutoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service  no usages  new *
public class ProdutoService {

    @Autowired  1 usage
    private ProdutoRepository produtoRepo;

    public List<ProdutoDTO> findAll(){  no usages  new *
        //retorna uma lista de ProdutoDTO
        return produtoRepo.findAll().stream()  Stream<Produto>
                .map(obj -> new ProdutoDTO(obj))  Stream<ProdutoD
                .collect(Collectors.toList());
    }
}
```

- Implemente a **ProdutoService.**
- **A camada Service** solicita a repository uma lista de Produto, converte para ProdutoDTO e retorna a lista de DTOs para a resource (camada que estarão nossos endpoints – api´s) que a solicitou.

Spring Boot

Criando a camada Resources Controller – findAll( )

PROF. ME. JEFFERSON PASSERINI

- A camada **Controller**, também conhecida como **Resource**, é uma parte fundamental da arquitetura de software que segue o modelo de Domain, Repository, Service e Resource.

- A principal responsabilidade da camada Controller é **gerenciar as requisições HTTP e enviar as respostas adequadas de volta ao cliente**, funcionando como a interface entre o usuário e a aplicação.

- Os Controllers (Resources) utilizam códigos HTTP para indicar claramente o resultado das requisições, facilitando a comunicação entre o cliente e o servidor e melhorando a experiência do usuário ao fornecer respostas precisas e significativas.

- **Funções:**

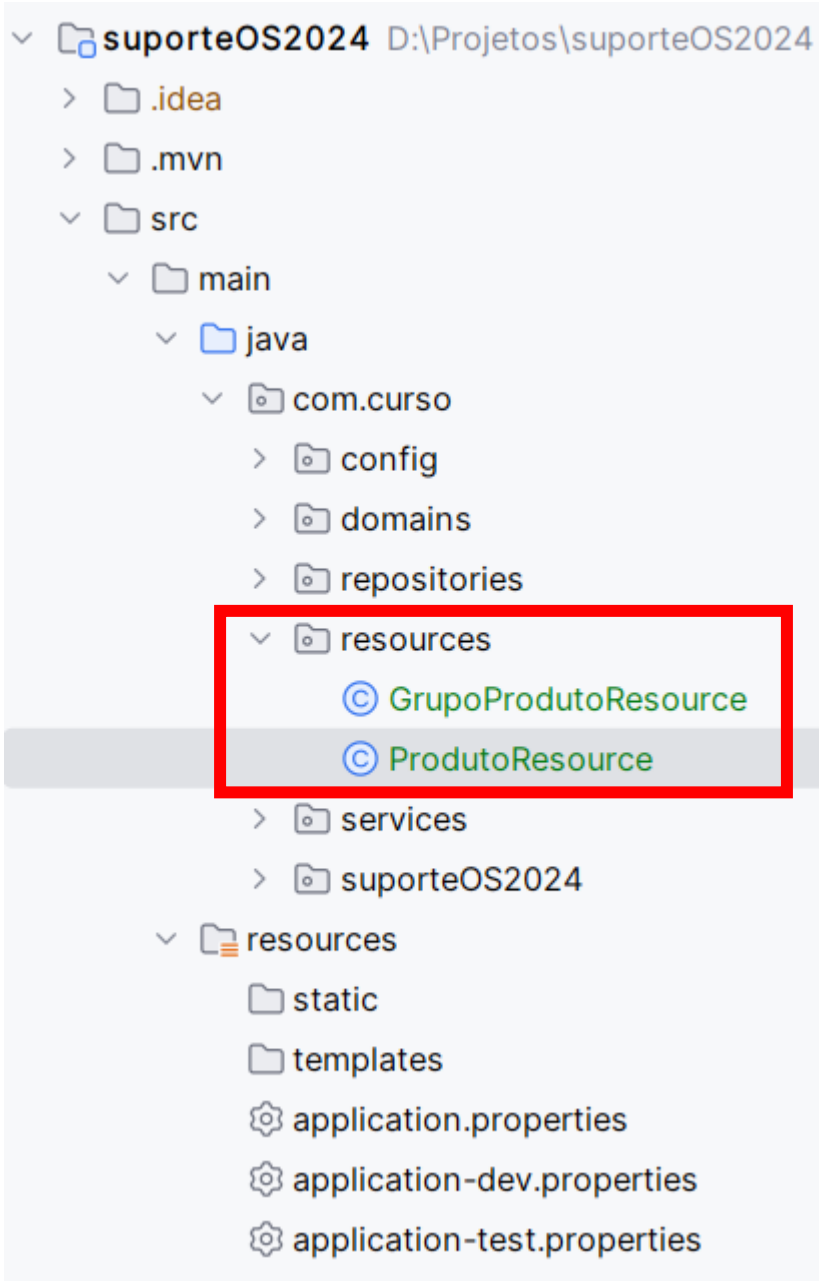
1. **Receber Requisições HTTP**: As classes Controller (Resources) são responsáveis por mapear e receber requisições HTTP (**GET, POST, PUT, DELETE**, etc.) vindas do cliente. Elas utilizam anotações ou decoradores específicos para mapear URLs para métodos de tratamento de requisições.
2. **Validar Dados de Entrada**: Antes de encaminhar os dados para a camada de serviço, os Controllers (Resources) realizam validações iniciais dos dados de entrada para garantir que estão no formato correto e que cumprem os requisitos básicos.
3. **Chamar Serviços**: Os Controllers (Resources) delegam a lógica de negócios para os serviços. Eles invocam métodos da camada Service para processar as requisições, mantendo-se livres de lógica de negócios complexa.
4. **Gerenciar Respostas HTTP**: Após receber o resultado dos serviços, os Controllers **(Resources)** formatam as respostas HTTP apropriadas (como status codes, headers e body) e as enviam de volta ao cliente. Isso pode incluir transformar dados de domínio em DTOs (Data Transfer Objects) antes de enviá-los.
5. **Tratamento de Exceções**: Os Controllers **(Resources)** também são responsáveis por capturar e tratar exceções que possam ocorrer durante o processamento das requisições, garantindo que respostas significativas sejam retornadas ao cliente mesmo em casos de erro.

- **Códigos HTTP – Códigos de Sucesso (2xx):**
- **200 OK:** ***Descrição:*** A requisição foi bem-sucedida e o servidor está retornando os dados solicitados. ***Exemplo:*** Uma requisição GET que retorna os detalhes de um pedido.
- **201 Created**: ***Descrição:*** A requisição foi bem-sucedida e resultou na criação de um novo recurso. O servidor deve retornar a localização do recurso criado no cabeçalho Location. ***Exemplo:*** Uma requisição POST que cria um novo pedido.
- **204 No Content**: Descrição: A requisição foi bem-sucedida, mas não há conteúdo para retornar. ***Exemplo:*** Uma requisição DELETE que remove um pedido.

- **Códigos HTTP – Códigos de Redirecionamento (3xx):**
- **301 Moved Permanently:** ***Descrição:*** O recurso solicitado foi movido permanentemente para uma nova URL.
- **302 Found:** ***Descrição*****:** O recurso solicitado foi encontrado em uma URL diferente.

- **Códigos HTTP – Códigos de Erro do Cliente (4xx):**
- **400 Bad Request:** ***Descrição:*** A requisição não pôde ser entendida pelo servidor devido a sintaxe incorreta ou dados inválidos. ***Exemplo:*** Dados de entrada inválidos em uma requisição POST.
- **401 Unauthorized:** ***Descrição:*** A requisição requer autenticação do usuário.
- **403 Forbidden:** ***Descrição*****:** O servidor entendeu a requisição, mas se recusa a autorizá-la.
- **404 Not Found:** ***Descrição:*** O recurso solicitado não foi encontrado no servidor. ***Exemplo:*** Tentativa de acessar um pedido que não existe
- **409 Conflict:** ***Descrição:*** A requisição não pôde ser completada devido a um conflito com o estado atual do recurso.

- **Códigos HTTP – Códigos de Erro do Servidor (5xx):**
- **500 Internal Server Error:** ***Descrição:*** Ocorreu um erro inesperado no servidor ao processar a requisição. ***Exemplo:*** Erro interno durante o processamento de uma requisição.
- **502 Bad Gateway:** ***Descrição:*** O servidor, ao atuar como gateway ou proxy, recebeu uma resposta inválida do servidor upstream.
- **503 Service Unavailable:** ***Descrição:*** O servidor está temporariamente incapaz de lidar com a requisição devido à sobrecarga ou manutenção.



- Crie um novo pacote em java/com/curso chamado **resources**.

- No pacote **resources** crie as classes java para Produto e GrupoProduto denominadas respectivamente: **ProdutoResource e GrupoProdutoResource** .

- Essas classes serão nossas **controllers** que publicarão nossos **endpoint REST**.

```java
package com.curso.resources;

import com.curso.domains.dtos.GrupoProdutoDTO;
import com.curso.services.GrupoProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController  no usages  new *
@RequestMapping(value = "/grupoproduto")
public class GrupoProdutoResource {

    @Autowired  1 usage
    private GrupoProdutoService grupoProdutoService;

    @GetMapping //exemplo http://localhost:8080/produto  no usages  new *
    public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(grupoProdutoService.findAll());
    }
}
```

- Vamos criar o endpoint que irá listar todos os GrupoProduto.
- A anotação **@RestController** indica que essa classe é um **controlador REST**, ou seja, ela gerencia as requisições HTTP e envia respostas no formato JSON (ou outros formatos) diretamente.
- **@RequestMapping(value="/grupoproduto") -** define a URL base para este controlador. Toda requisição que tiver **/grupoproduto** no caminho será tratada por este controlador. Por exemplo, uma requisição para **http://localhost:8080/grupoproduto** será roteada para essa classe.

```java
package com.curso.resources;

import com.curso.domains.dtos.GrupoProdutoDTO;
import com.curso.services.GrupoProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController  no usages  new *
@RequestMapping(value = "/grupoproduto")
public class GrupoProdutoResource {

    @Autowired  1 usage
    private GrupoProdutoService grupoProdutoService;

    @GetMapping //exemplo http://localhost:8080/produto  no usages  new *
    public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(grupoProdutoService.findAll());
    }
}
```

- A anotação **@Autowired** para que o Spring inicie automaticamente (injeção de dependência) o objeto **grupoProdutoService**, que contém a lógica de negócio da aplicação. O Spring cuida da criação e gerenciamento desse serviço, o que ajuda a manter o código desacoplado.

```java
package com.curso.resources;

import com.curso.domains.dtos.GrupoProdutoDTO;
import com.curso.services.GrupoProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController  no usages  new *
@RequestMapping(value = "/grupoproduto")
public class GrupoProdutoResource {

    @Autowired  1 usage
    private GrupoProdutoService grupoProdutoService;

    @GetMapping //exemplo http://localhost:8080/produto  no usages  new *
    public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(grupoProdutoService.findAll());
    }
}
```

- O método **findAll()** é a parte principal que responde às requisições **HTTP GET para a URL /grupoproduto**.

Explicando cada parte:

@GetMapping: essa anotação define que este método responde a **requisições do tipo GET** (usadas para buscar dados).

ResponseEntity<List<GrupoProdutoDTO>>: o retorno do método é um **ResponseEntity**, que encapsula a **resposta HTTP.** Ele contém uma lista de objetos GrupoProdutoDTO.

grupoProdutoService.findAll(): aqui é chamado o serviço **GrupoProdutoService**, especificamente o **método findAll(),** que recupera todos os grupos de produtos.

ResponseEntity.ok().body(...): ResponseEntity.ok() cria uma **resposta HTTP** com o código **de status 200 (OK),** e **body(...)** define o conteúdo da resposta, que será a lista de grupos de produtos.

Criando Projeto Spring Boot – Resources

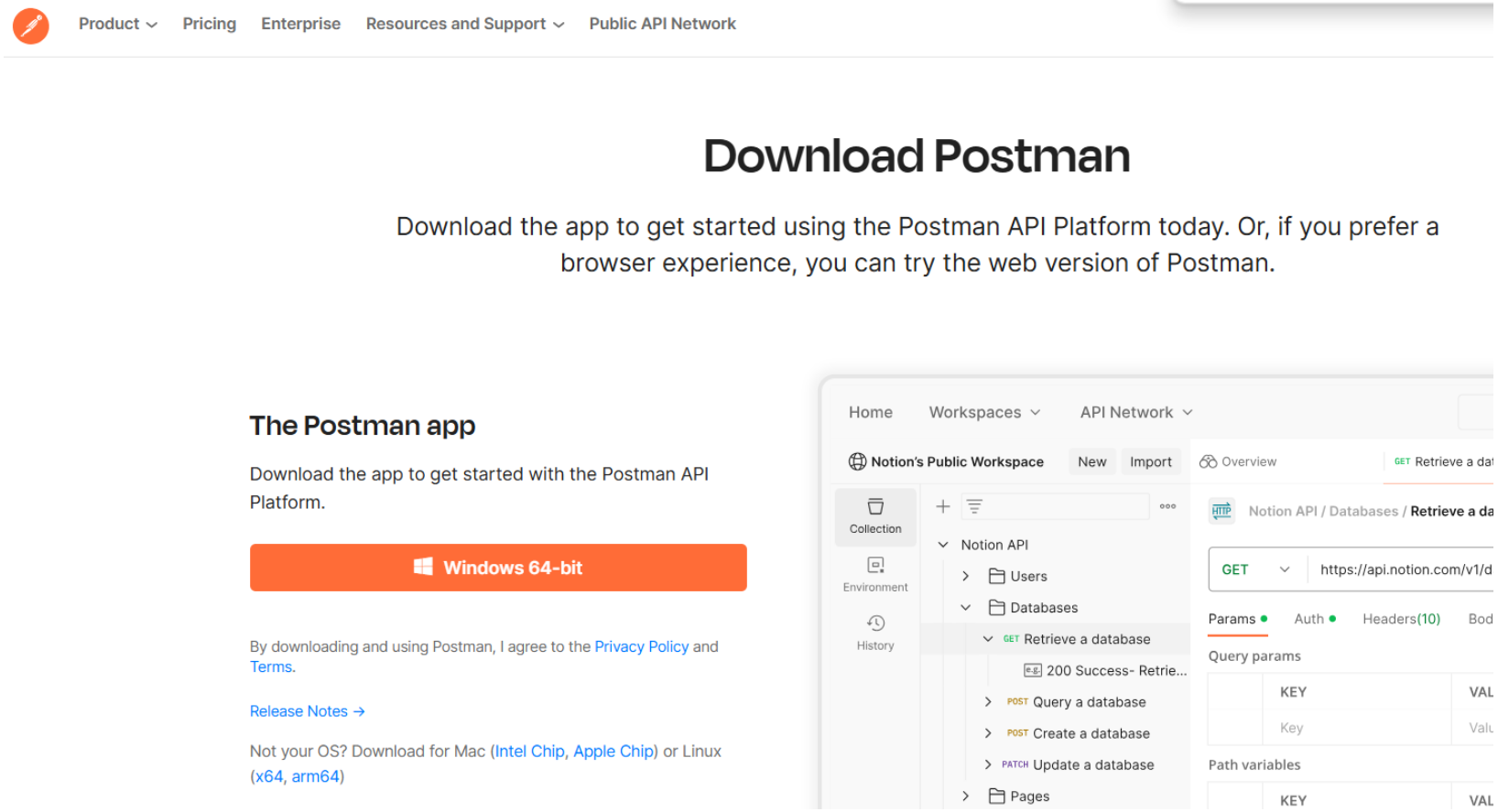
- **Agora vamos criar o endpoint que irá listar todos os Produtos.**

```java
package com.curso.resources;

import com.curso.domains.dtos.ProdutoDTO;
import com.curso.services.ProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController  no usages  new *
@RequestMapping(value = "/produto")
public class ProdutoResource {

    @Autowired  1 usage
    private ProdutoService produtoService;

    @GetMapping //exemplo http://localhost:8080/produto  no usages  ne
    public ResponseEntity<List<ProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(produtoService.findAll());
    }
}
```

Spring Boot

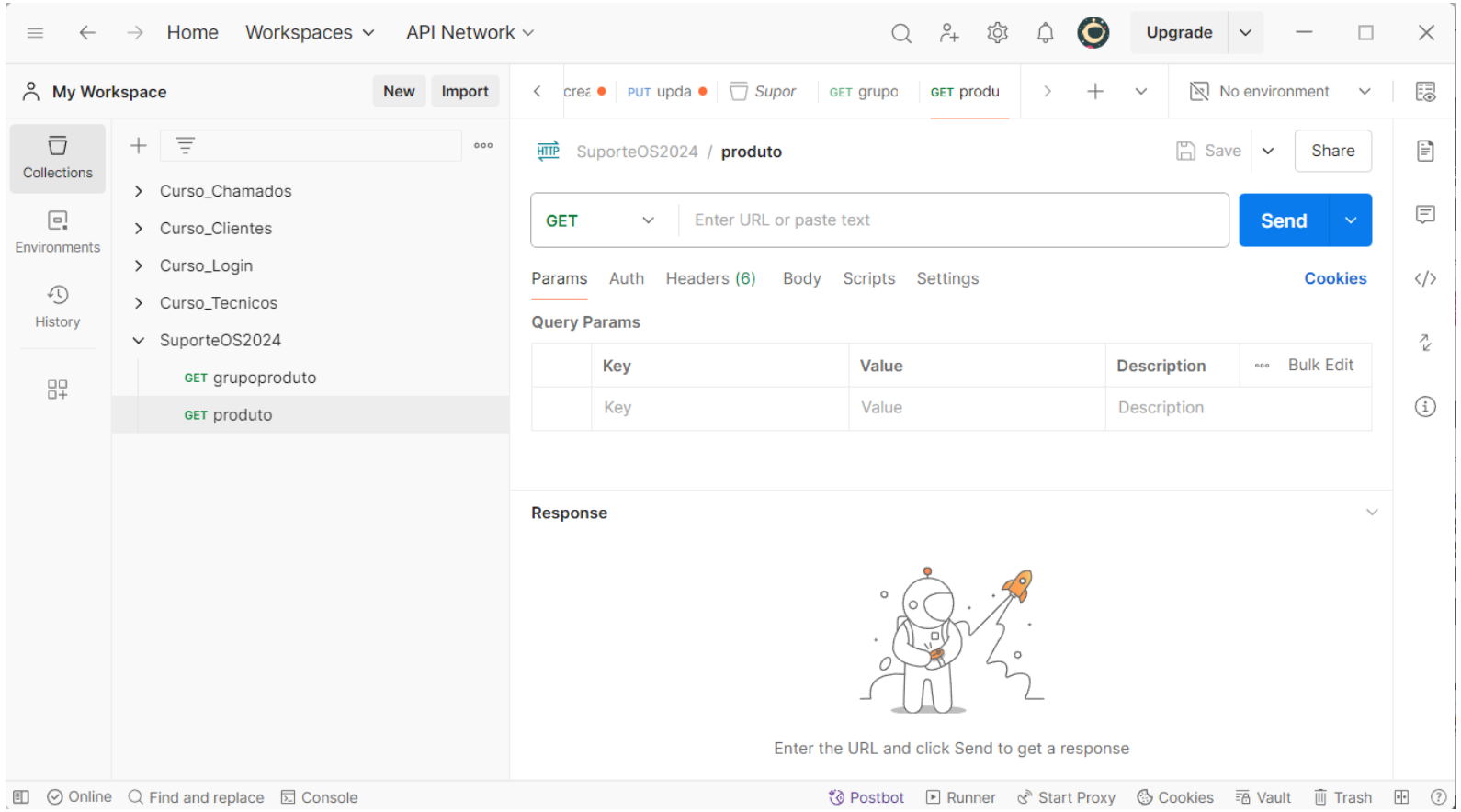
Testando nossos endpoit´s findAll( )

PROF. ME. JEFFERSON PASSERINI



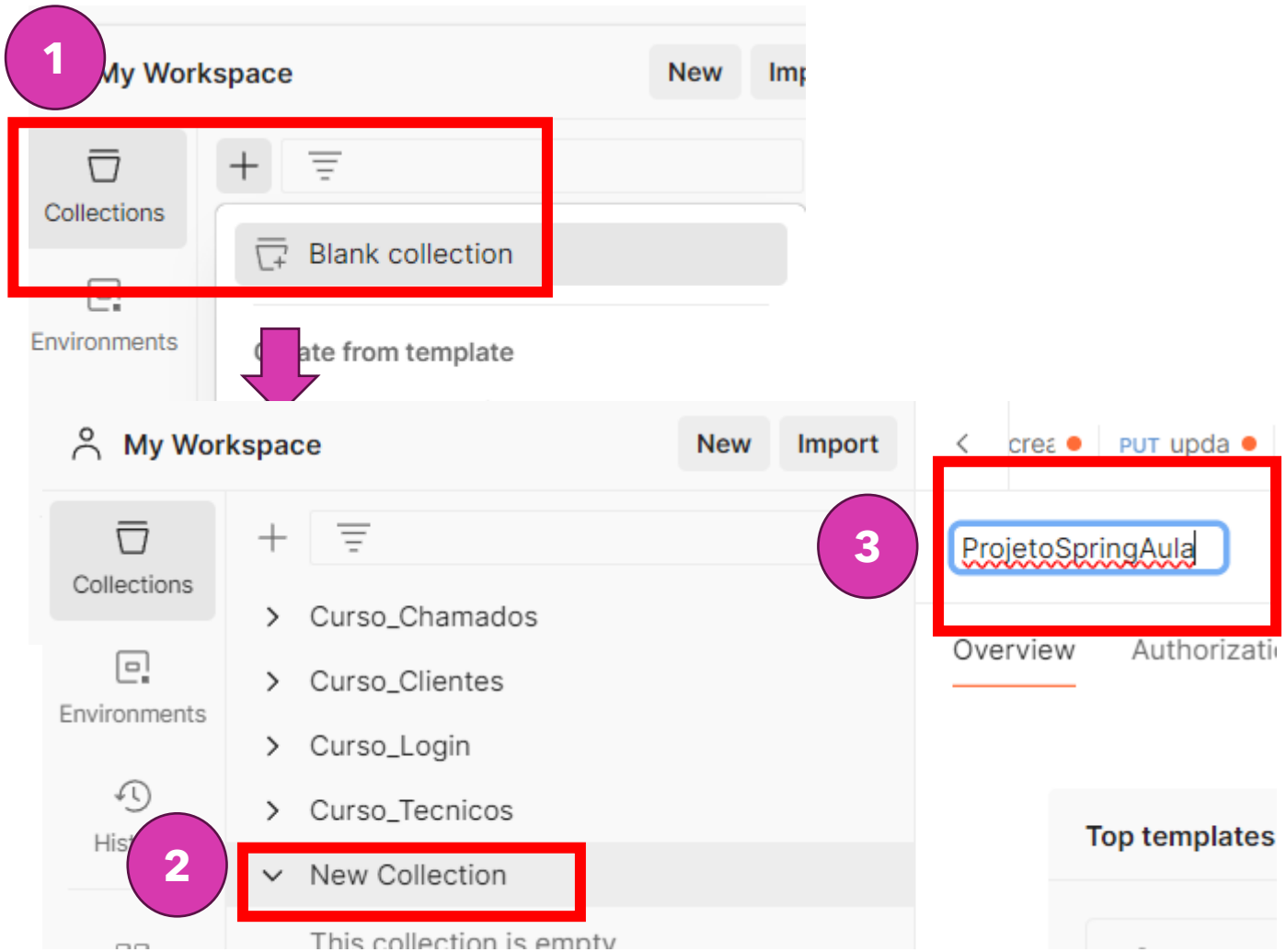
- Faça o dowload do Postman → www.postman.com/downloads
- Faça sua instalação.

Criando Projeto Spring Boot – Teste



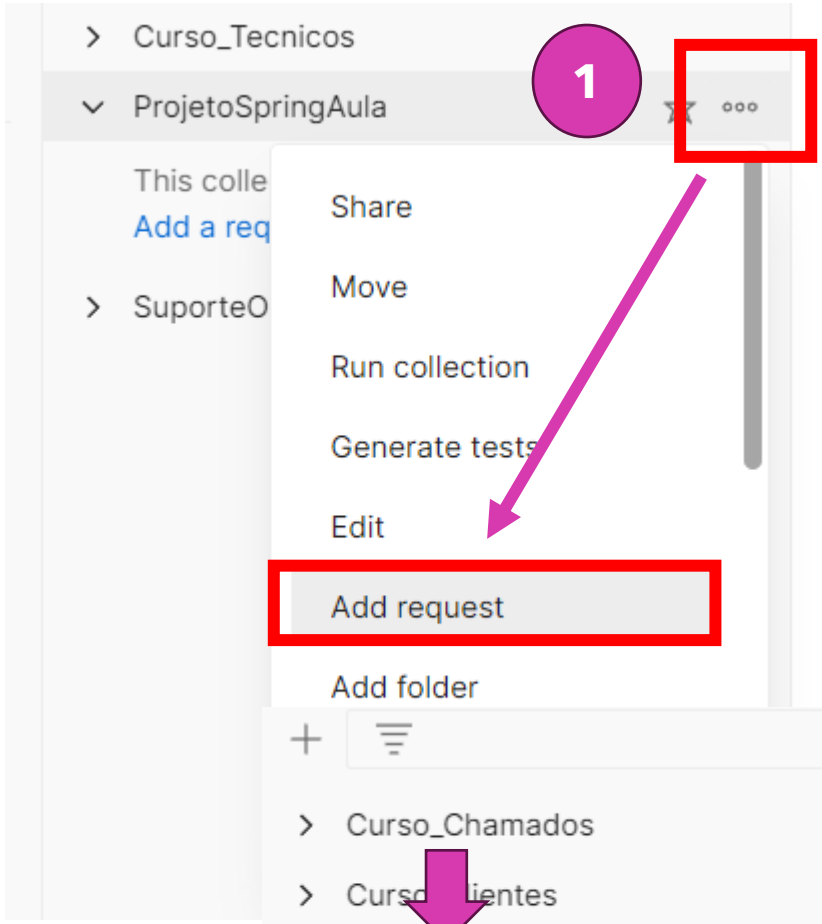
- Quando finalizar a instalação faça login.
- Se não tiver usuário criado crie um novo usuário pois você terá benefícios de manter seus endpoints vinculados ao seu usuário.

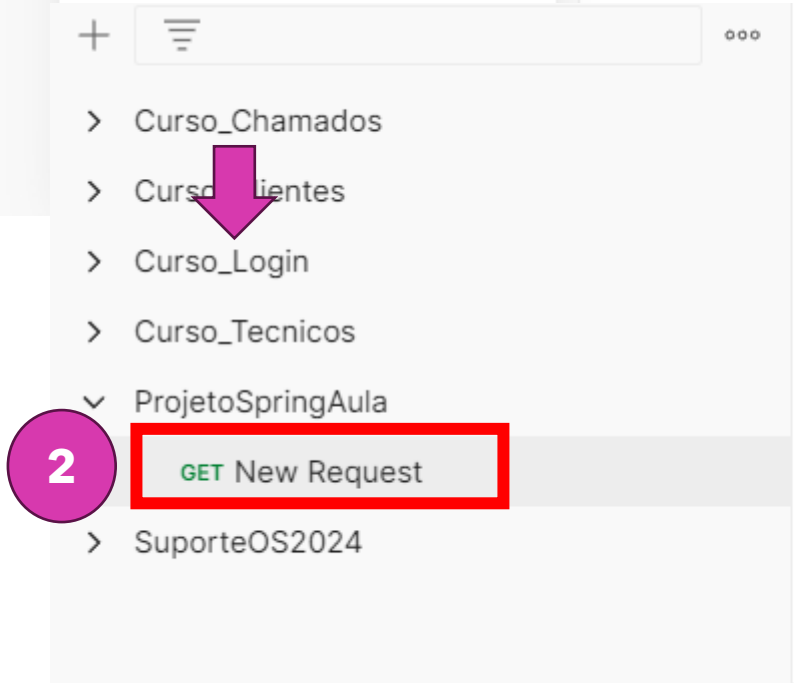
Criando Projeto Spring Boot – Teste

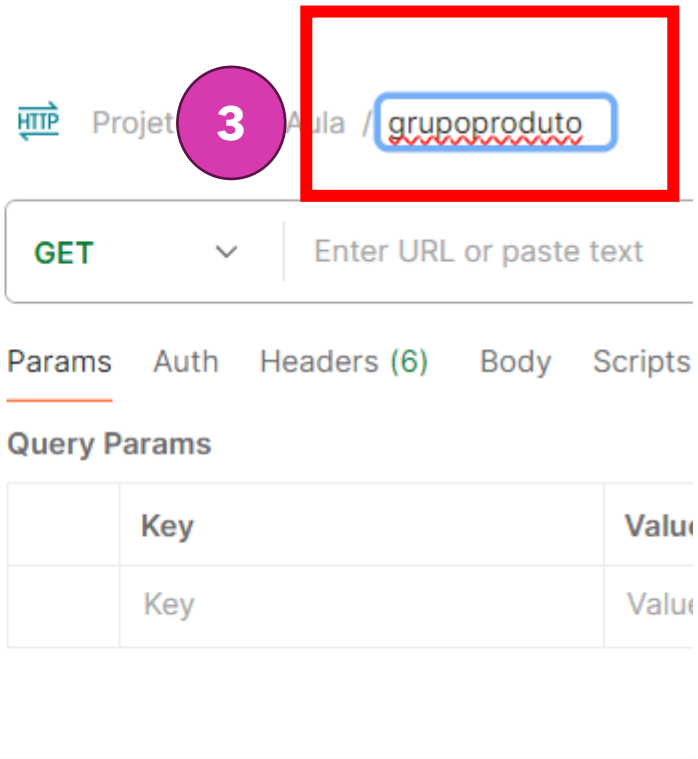


- 1 – Crie uma nova coleção para o seu projeto.
- 2 – Nova coleção criada
- 3 – Altere o nome da coleção conforme indicado.

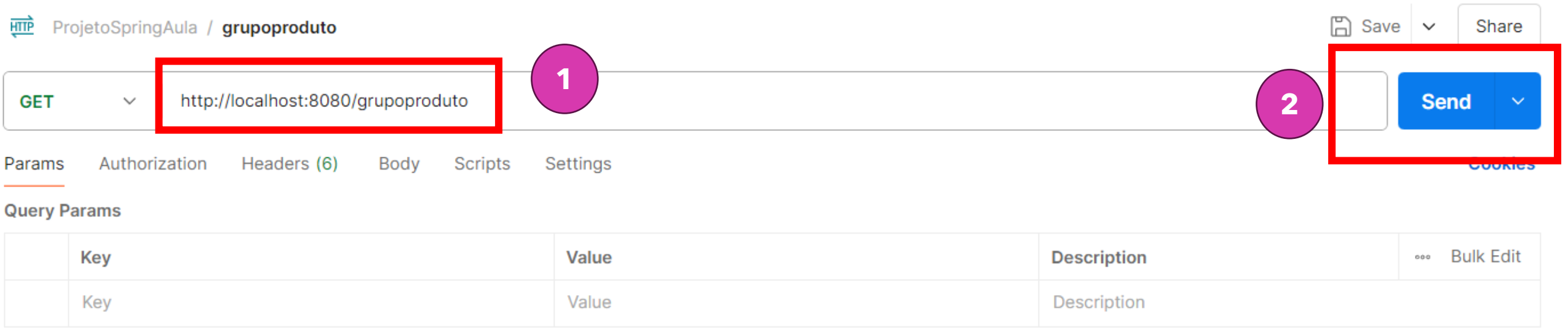
Criando Projeto Spring Boot – Teste



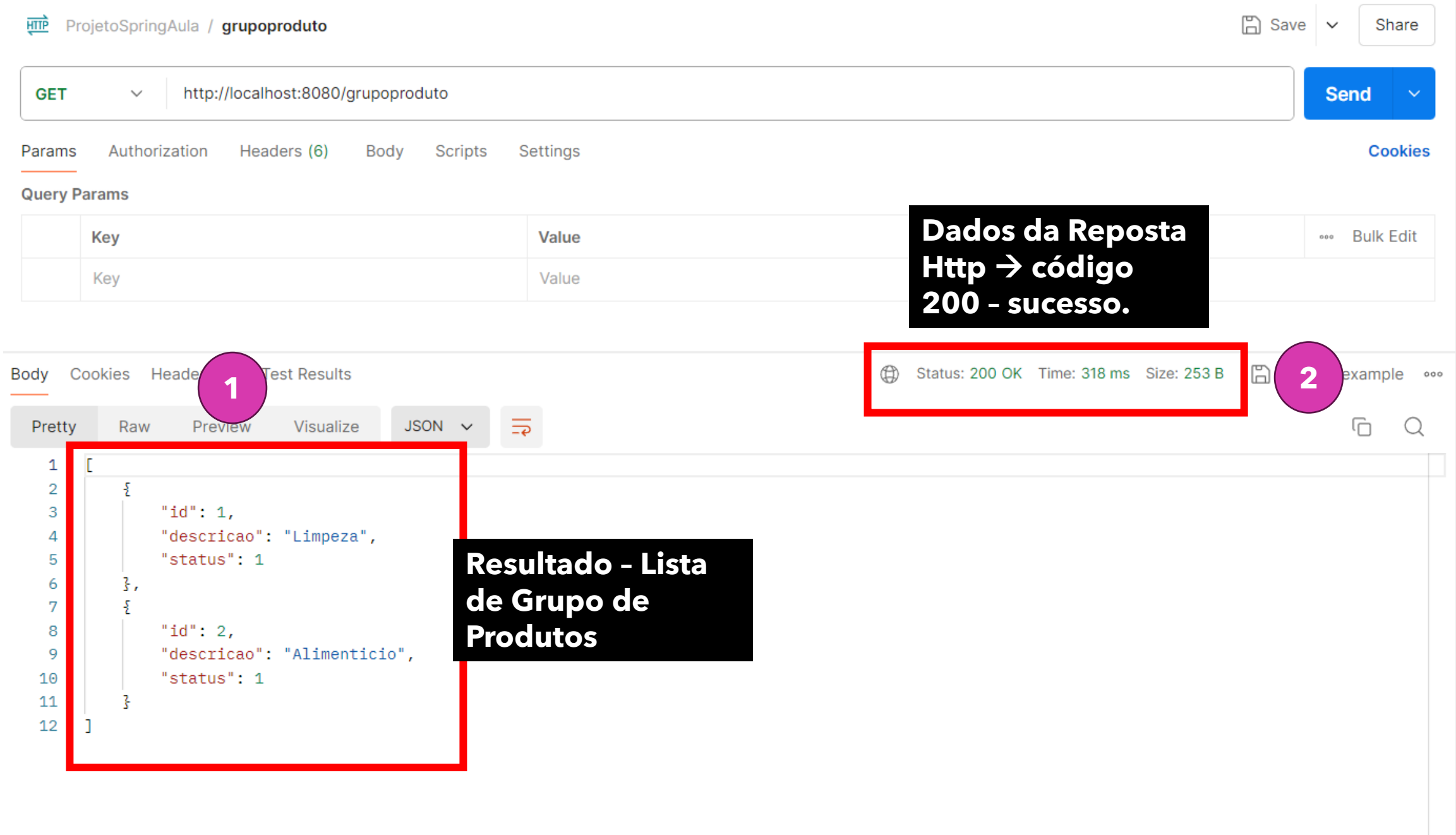


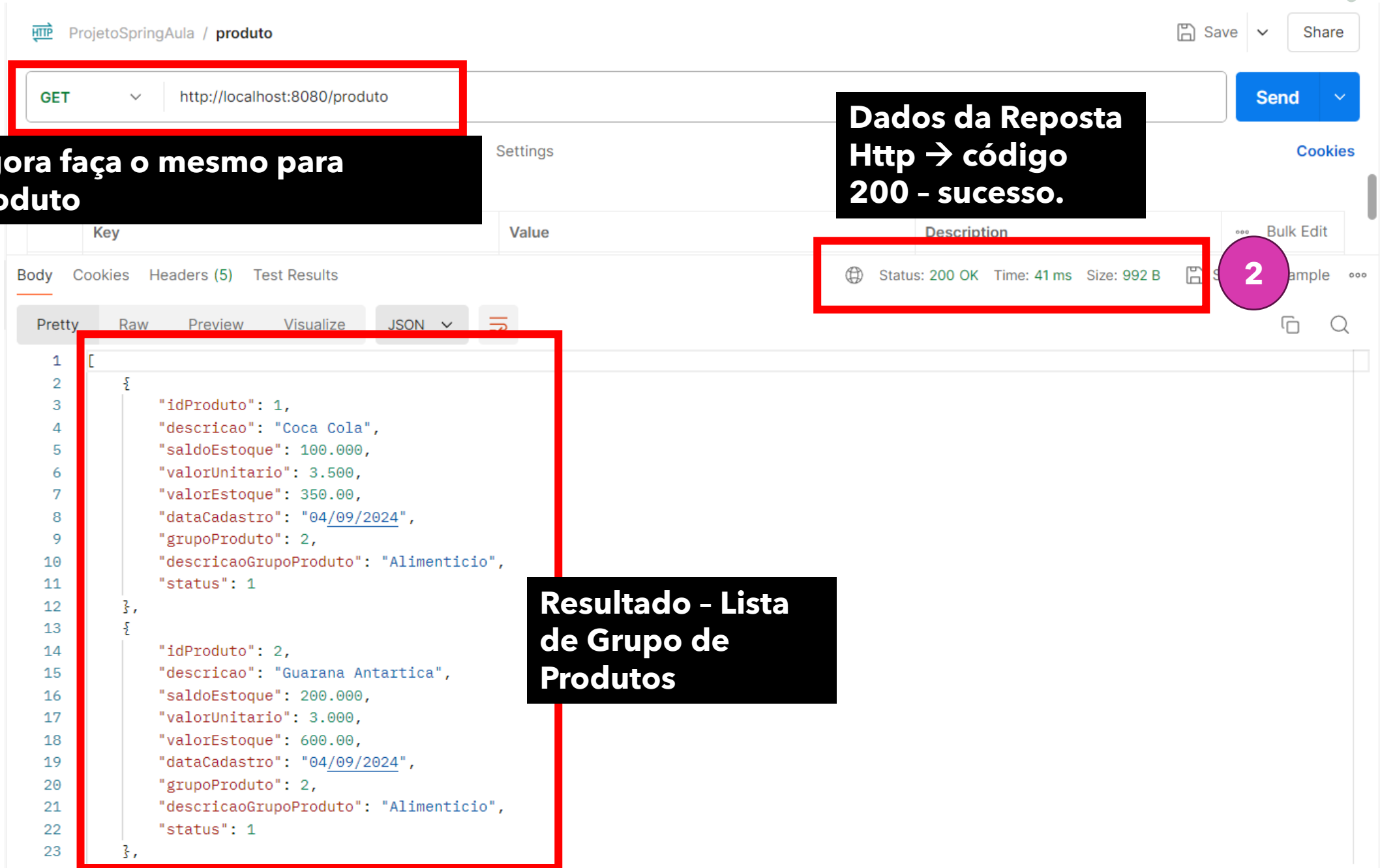


- 1 – Clique no botão de "..." na coleção que você acabou de criar e escolha a opção Add Request.
- 2 – Foi criada uma nova requisição do tipo **GET**
- 3 – Altere o endereço da requisição para **grupoproduto** como demonstrado.



- 1 – Coloque o endereço para o seu endpoint, observe que estamos realizado uma requisição GET como é o caso do findAll()
- 2 – Clique no botão SEND.
- **Não esqueça que seu projeto Java Spring deve estar rodando.**


ProjetoSpringAula / grupoproduto
Save
Share
GET
http://localhost:8080/grupoproduto
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Key
Value
Dados da Reposta Http → código 200 - sucesso.
Body
Cookies
Heade
1
Test Results
Status: 200 OK
Time: 318 ms
Size: 253 B
2
example
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"descricao": "Limpeza",
"status": 1
},
{
"id": 2,
"descricao": "Alimenticio",
"status": 1
}
]
Resultado - Lista de Grupo de Produtos




```
[
    {
        "idProduto": 1,
        "descricao": "Coca Cola",
        "saldoEstoque": 100.000,
        "valorUnitario": 3.500,
        "valorEstoque": 350.00,
        "dataCadastro": "04/09/2024",
        "grupoProduto": 2,
        "descricaoGrupoProduto": "Alimenticio",
        "status": 1
    },
    {
        "idProduto": 2,
        "descricao": "Guarana Antartica",
        "saldoEstoque": 200.000,
        "valorUnitario": 3.000,
        "valorEstoque": 600.00,
        "dataCadastro": "04/09/2024",
        "grupoProduto": 2,
        "descricaoGrupoProduto": "Alimenticio",
        "status": 1
    },
```