



# Spring Boot

## Correções no Projeto

- Nas definições iniciais de nosso projeto nas classes GrupoProduto, GrupoProdutoDTO, Produto e ProdutoDTO, **definimos** respectivamente os **ids** destas classes com os **tipos primários int e long**.

```
7 public class GrupoProduto { 2 usages new *  
8  
9 private int id; 6 usages  
10 private String descricao; 6 usages  
11 private Status status; 3 usages  
12
```

- Agora como **vamos implementar o create** para inserirmos os dados de ambas classes o **ideal que os campos ids estejam valendo null para que o JPA faça o processo de geração da chave primário junto do SGBD**.

```
7 public class Produto { no usages new *  
8  
9 private long idProduto; no usages  
10 private String descricao; no usages  
11 private double saldoEstoque; no usages  
12 private double valorUnitario; no usages  
13 private LocalDate dataCadastro; no usages  
14 private Status status; no usages  
15  
16 }  
17
```

- Só que os tipos primário int e long não aceitam valores nulos. Vamos resolver isso!**

- Vamos alterar o valor primários para as respectivas classes que as representam:
- Em **GrupoProduto** e **GrupoProdutoDTO**, altere o tipo do campo **de int para Integer**.
- E em **Produto** e **ProdutoDTO** altere **de long para Long**.
- **Essas alterações gerarão reflexos nos construtores e métodos Getters e Setters dessas classes, faça as correções trocando os tipos de dados como fizemos na definição dos atributos.**

```
7 public class GrupoProduto { 2 usages new *
8
9 private int id; 6 usages
10 private String descricao; 6 usages
11 private Status status; 3 usages
12
```

```
7 public class Produto { no usages new *
8
9 private long idProduto; no usages
10 private String descricao; no usages
11 private double saldoEstoque; no usages
12 private double valorUnitario; no usages
13 private LocalDate dataCadastro; no usages
14 private Status status; no usages
15
16 }
17
```

- Outro local que devemos fazer correções é na classe DBService, que faz a carga inicial de dados do sistema.
- Agora podemos passe null nos ids dos construtores.

```
@Service 4 usages jeffersonarpasserini *  
public class DBService {
```

```
@Autowired  
private GrupoProdutoRepository grupoProdutoRepo;
```

```
@Autowired  
private ProdutoRepository produtoRepo;
```

```
public void initDB() { 2 usages jeffersonarpasserini *
```

```
GrupoProduto grupo01 = new GrupoProduto(id: null, descricao: "Limpeza", Status.ATIVO);  
GrupoProduto grupo02 = new GrupoProduto(id: null, descricao: "Alimenticio", Status.ATIVO);
```

```
Produto produto01 = new Produto(idProduto: null, codigoBarra: "1111", descricao: "Coca Cola", new BigDecimal(val: "100"), new BigDecimal(val: "3.5"),  
    LocalDate.now(), grupo02, Status.ATIVO);  
Produto produto02 = new Produto(idProduto: null, codigoBarra: "2222", descricao: "Guarana Antartica", new BigDecimal(val: "200"), new BigDecimal(val: "3.0"),  
    LocalDate.now(), grupo02, Status.ATIVO);  
Produto produto03 = new Produto(idProduto: null, codigoBarra: "3333", descricao: "Detergente Limpol", new BigDecimal(val: "300"), new BigDecimal(val: "4.0"),  
    LocalDate.now(), grupo01, Status.ATIVO);  
Produto produto04 = new Produto(idProduto: null, codigoBarra: "4444", descricao: "Sabão em Pó OMO", new BigDecimal(val: "400"), new BigDecimal(val: "15.5"),  
    LocalDate.now(), grupo02, Status.ATIVO);
```



# Spring Boot Endpoint Create - GrupoProduto



- Vamos criar um Endpoint Create() para inserir novos gruposprodutos e produtos em nosso banco de dados.
- **Pela regra a camada Resources trabalhamos com o padrão DTO → GrupoProdutoDTO e ProdutoDTO.**
- **Então essa camada irá enviar para a Service → GrupoProdutoService ou ProdutoService um objeto DTO que deverá converter para o padrão do domínio.**
- Então em nossa GrupoProduto e na Produto em Domain vamos criar um novo construtor que irá se encarregar desta conversão DTO → Domain

```
32  
33 > public GrupoProduto() { this.status = Status.ATIVO; }  
36  
37 @a public GrupoProduto(int id, String descricao, Status status) {  
38     this.id = id;  
39     this.descricao = descricao;  
40     this.status = status;  
41 }  
42  
43 @ public GrupoProduto(GruoProdutoDTO dto) { no usages new *  
44     this.id = dto.getId();  
45     this.descricao = dto.getDescricao();  
46     this.status = Status.toEnum(dto.getStatus());  
47 }  
48
```

- **Implemente no novo construtor em GrupoProduto na camada Domain como demonstrado.**

## Criando Projeto Spring Boot – Create

- Agora vamos criar na Service (**GrupoProdutoService**) o método responsável por receber os dados (DTO) da camada **Resources** (Controllers).
- Esse método deverá converter o padrão de dados de **DTO para Domain** e solicitar a camada **Repository** que faça a gravação através do método **SAVE()**.
- **Implemente no novo método em GrupoProdutoService na camada Service como demonstrado.**

```
1 package com.curso.services;
2
3 > import ...
13
14 @Service 2 usages 1 jeffersonarpasserini +1 *
15 public class GrupoProdutoService {
16
17     @Autowired
18     private GrupoProdutoRepository grupoProdutoRepo;
19
20     public List<GrupoProdutoDTO> findAll(){ 1 usage 1 jeffersonarpasserini
21         //retorna uma lista de ProdutoDTO
22         return grupoProdutoRepo.findAll().stream() Stream<GrupoProduto
23             .map(obj -> new GrupoProdutoDTO(obj)) Stream<GrupoProd
24             .collect(Collectors.toList());
25     }
26
27     public GrupoProduto findById(int id){ 1 usage 1 jeffersonarpasserini
28         Optional<GrupoProduto> obj = grupoProdutoRepo.findById(id);
29         return obj.orElse( other: null);
30     }
31
32 @ public GrupoProduto create(GrupoProdutoDTO dto){ new *
33     dto.setId(null);
34     GrupoProduto obj = new GrupoProduto(dto);
35     return grupoProdutoRepo.save(obj);
36 }
37 }
```

## Criando Projeto Spring Boot – Create

- Na camada **Resources** → **GrupoProdutoResource** vamos implementar o método **create**.
- Este método é do tipo **PostMapping** → requisição Post do protocolo Http.
- Esse método receberá o **conteúdo que virá no corpo da requisição Http**, assim utilizamos como parâmetro o **@RequestBody**.
- Implemente no novo método em **GrupoProdutoResource** na camada Resource como demonstrado

```
17 @RestController @jeffersonarpasserini *
18 @RequestMapping(value = "/grupoproduto")
19 public class GrupoProdutoResource {
20
21     @Autowired
22     private GrupoProdutoService grupoProdutoService;
23
24     @GetMapping //exemplo http://localhost:8080/produto @jeffersonarpasserini
25     public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
26         return ResponseEntity.ok().body(grupoProdutoService.findAll());
27     }
28
29     @GetMapping(value =("/{id}") //exemplo http://localhost:8080/grupoproduto/1 @jeffersonarpasserini
30     public ResponseEntity<GrupoProdutoDTO> findById(@PathVariable Integer id){
31         GrupoProduto obj = this.grupoProdutoService.findById(id);
32         return ResponseEntity.ok().body(new GrupoProdutoDTO(obj));
33     }
34
35     @PostMapping new *
36     public ResponseEntity<GrupoProdutoDTO> create(@RequestBody GrupoProdutoDTO dto) {
37         GrupoProduto grupoProduto = grupoProdutoService.create(dto);|
38         // Cria o URI para o recurso criado
39         URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
40             .buildAndExpand(grupoProduto.getId()).toUri();
41         // Retorna a resposta com o status 201 Created e o local do recurso criado
42         return ResponseEntity.created(uri).build();
43     }
44 }
```




## Criando Projeto Spring Boot – Create

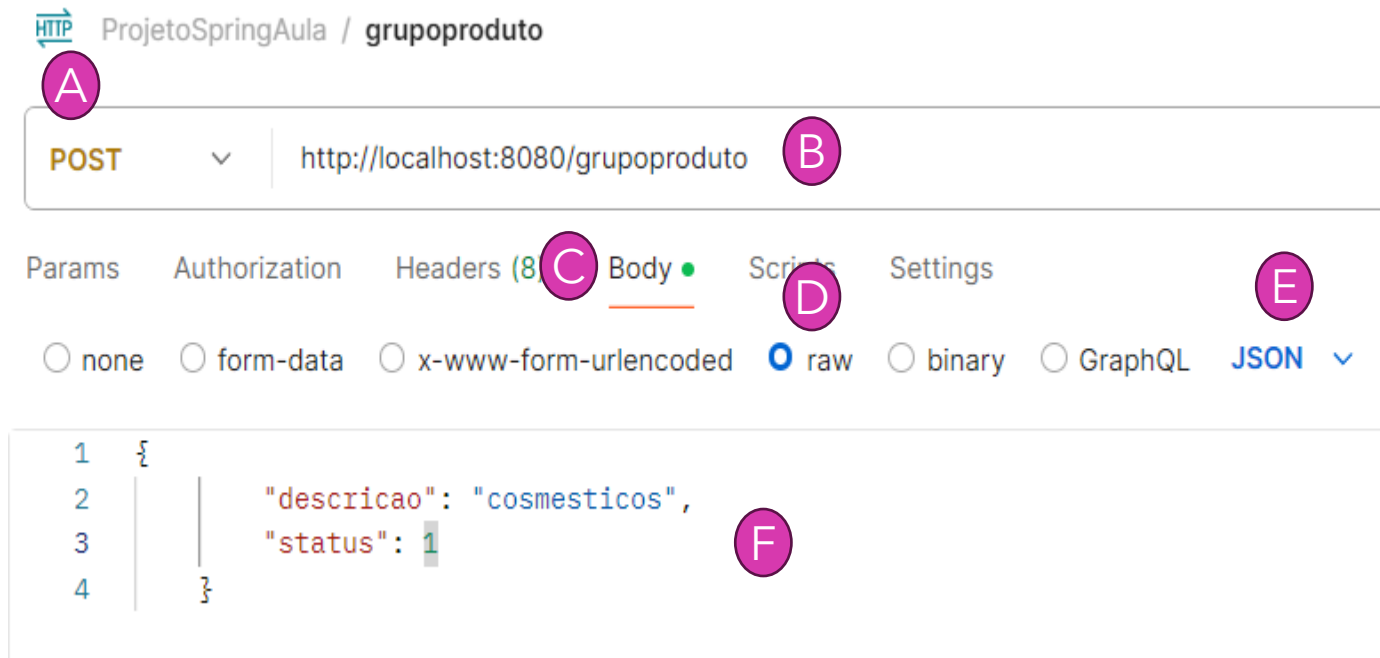
- Na linha 37 da imagem temos a criação de um objeto de domínio GrupoProduto a partir do DTO.
- E como padrão de resposta devemos retornar a URI de acesso ao novo grupo de produto que foi cadastrado.
- Assim nas linhas 39 e 40 temos um construtor de URI (**ServletUriComponentBuilder**) para retornarmos na linha 42.

```
17 @RestController @jeffersonarpasserini *
18 @RequestMapping(value = "/grupoproduto")
19 public class GrupoProdutoResource {
20
21     @Autowired
22     private GrupoProdutoService grupoProdutoService;
23
24     @GetMapping //exemplo http://localhost:8080/produto @jeffersonarpasserini
25     public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
26         return ResponseEntity.ok().body(grupoProdutoService.findAll());
27     }
28
29     @GetMapping(value =("/{id}") //exemplo http://localhost:8080/grupoproduto/1 @jeffersonarpasserini
30     public ResponseEntity<GrupoProdutoDTO> findById(@PathVariable Integer id){
31         GrupoProduto obj = this.grupoProdutoService.findById(id);
32         return ResponseEntity.ok().body(new GrupoProdutoDTO(obj));
33     }
34
35     @PostMapping new *
36     public ResponseEntity<GrupoProdutoDTO> create(@RequestBody GrupoProdutoDTO dto) {
37         GrupoProduto grupoProduto = grupoProdutoService.create(dto);|
38         // Cria o URI para o recurso criado
39         URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
40             .buildAndExpand(grupoProduto.getId()).toUri();
41         // Retorna a resposta com o status 201 Created e o local do recurso criado
42         return ResponseEntity.created(uri).build();
43     }
44 }
```

- Para testarmos nosso endpoint no Postman vamos copiar um Json de nosso GrupoProduto a partir de uma consulta que já implementamos em nosso projeto.
- Não precisaremos de todos os dados para inserir um novo grupo produto. Não são necessários: **id**.



```
{  
  "id": 1,  
  "descricao": "Limpeza",  
  "status": 1  
},
```



POST http://localhost:8080/grupoproduto Send

Params Auth Headers (8) Body Scripts Settings

raw JSON

```
1 {
2   "descricao": "cosmesticos",
3   "status": 1
4 }
```

A

Body Cookies Headers (5) Test Results

201 Created • 57 ms • 176 B

Key	Value
Location	http://localhost:8080/grupoproduto/3
Content-Length	0
Date	Thu, 24 Oct 2024 00:15:05 GMT
Keep-Alive	timeout=60
Connection	keep-alive

B

GET http://localhost:8080/grupoproduto/3 C

Params Auth Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "descricao": "cosmesticos",
4   "status": 1
5 }
```

D

- Para verificar a resposta de nosso endpoint escolha Headers (A) em (B) observamos a URL gerada a partir da resposta do endpoint.
- Copie e cole essa URL em uma requisição do tipo GET no Postman (C) e irá retornar os dados gravados (D).



# Spring Boot Endpoint Create - Produto

```
@NotNull 5 usages
@Digits(integer = 15, fraction = 3)
private BigDecimal saldoEstoque;

@NotNull 6 usages
@Digits(integer = 15, fraction = 3)
private BigDecimal valorUnitario;

@NotNull 5 usages
@Digits(integer = 15, fraction = 2)
private BigDecimal valorEstoque;
```

- Se observarmos a definição dos campos de saldoEstoque, valorUnitario verifica-se que possuem 3 casas decimais.
- Já o campo valorEstoque possui apenas 2 casas decimais, este campo é obtido a partir de um cálculo (multiplicação dos atributos anteriores) que é realizado com 3 casas decimais.
- Isso pode causar um erro no JPA pois ele irá persistir o campo calculado com 3 casas decimais que na verdade deveriam ser duas.



- **Vamos corrigir o construtor de Produto**

```
public Produto(Long idProduto, String codigoBarra, String descricao, BigDecimal saldoEstoque, BigDecimal valorUnitario,
               LocalDate dataCadastro, GrupoProduto grupoProduto, Status status) {
    this.idProduto = idProduto;
    this.codigoBarra = codigoBarra;
    this.descricao = descricao;
    this.saldoEstoque = saldoEstoque;
    this.valorUnitario = valorUnitario;
    this.dataCadastro = dataCadastro;
    this.grupoProduto = grupoProduto;
    this.status = status;
    this.valorEstoque = saldoEstoque.multiply(valorUnitario)
        .setScale(newScale: 2, BigDecimal.ROUND_HALF_UP);
}
```

- **Na multiplicação o método setScale() junto com o parâmetro BigDecimal.ROUND\_HALF\_UP garantirá o correto arredondamento para 2 casas decimais.**

- Agora assim como fizemos em GrupoProduto vamos implementar no novo construtor para converter DTO para domínio em Produto na camada Domain como demonstrado.

```
public Produto(ProdutoDTO dto) { 3 usages new *
    this.idProduto = dto.getIdProduto();
    this.codigoBarra = dto.getCodigoBarra();
    this.descricao = dto.getDescricao();
    this.valorUnitario = dto.getValorUnitario();
    this.saldoEstoque = dto.getSaldoEstoque();
    this.dataCadastro = dto.getDataCadastro();
    this.status = Status.toEnum(dto.getStatus());;
    //atribuição do grupoProduto
    this.grupoProduto = new GrupoProduto();
    this.grupoProduto.setId(dto.getGrupoProduto());
    //arredonda para baixo e para cima,
    // se estiver exatamente no meio arredonda para cima
    this.valorEstoque = dto.getSaldoEstoque().multiply(valorUnitario)
        .setScale(newScale: 2, BigDecimal.ROUND_HALF_UP);
}
```

## Criando Projeto Spring Boot – Create

- Agora vamos criar na Service (**ProdutoService**) o método responsável por receber os dados (DTO) da camada **Resources** (Controllers).
- Esse método deverá converter o padrão de dados de **DTO para Domain** e solicitar a camada **Repository** que faça a gravação através do método **SAVE()**.
- **Implemente no novo método em ProdutoService na camada Service como demonstrado.**

```
15
16 @Service 4 usages 1 jeffersonarpasserini +1 *
17 public class ProdutoService {
18
19     @Autowired
20     private ProdutoRepository produtoRepo;
21
22     public List<ProdutoDTO> findAll(){ 1 usage 1 jeffersonarpasserini *
23         //retorna uma lista de ProdutoDTO
24         return produtoRepo.findAll().stream()
25             .map(obj -> new ProdutoDTO(obj)).collect(Collectors.toList());
26     }
27
28     public Produto findById(Long id){ 1 usage 1 jeffersonarpasserini
29         Optional<Produto> obj = produtoRepo.findById(id);
30         return obj.orElseThrow (() -> new ObjectNotFoundException("Produto não encontrado! Id: "+id));
31     }
32
33     public Produto findByIdByCodigoBarra(String codigoBarra){ 1 usage 1 jeffersonarpasserini
34         Optional<Produto> obj = produtoRepo.findByIdByCodigoBarra(codigoBarra);
35         return obj.orElseThrow(
36             () -> new ObjectNotFoundException("Produto não encontrado! CodigoBarra: "+codigoBarra)
37         );
38     }
39
40     public Produto create(ProdutoDTO dto){ new *
41         dto.setIdProduto(null);
42         Produto obj = new Produto(dto);
43         return produtoRepo.save(obj);
44     }
45
46 }
```

## Criando Projeto Spring Boot – Create

- Na camada **Resources** → **ProdutoResource** vamos implementar o método **create**.
- Este método é do tipo **PostMapping** → requisição Post do protocolo Http.
- **Esse método receberá o conteúdo que virá no corpo da requisição Http**, assim utilizamos como parâmetro o **@RequestBody**.
- **Implemente no novo método em ProdutoResource na camada Resource como demonstrado**

```
18 public class ProdutoResource {
```

```
20 @Autowired
```

```
21 private ProdutoService produtoService;
```

```
23 @GetMapping //exemplo http://localhost:8080/produto jeffersonarpasserini
```

```
24 public ResponseEntity<List<ProdutoDTO>> findAll(){
```

```
25     return ResponseEntity.ok().body(produtoService.findAll());
```

```
26 }
```

```
28 @GetMapping(value = "{id}") //exemplo http://localhost:8080/produto/1 jeffersonarpasserini
```

```
29 public ResponseEntity<ProdutoDTO> findById(@PathVariable Long id){
```

```
30     Produto obj = this.produtoService.findById(id);
```

```
31     return ResponseEntity.ok().body(new ProdutoDTO(obj));
```

```
32 }
```

```
34 @GetMapping(value = "/codigobarra/{codigoBarra}") //exemplo http://localhost:8080/produto/codigobarra/1 jeffersonarpasserini
```

```
35 public ResponseEntity<ProdutoDTO> findById(@PathVariable String codigoBarra){
```

```
36     Produto obj = this.produtoService.findbyCodigoBarra(codigoBarra);
```

```
37     return ResponseEntity.ok().body(new ProdutoDTO(obj));
```

```
38 }
```

```
39 @PostMapping new *
```

```
40 public ResponseEntity<ProdutoDTO> create(@RequestBody ProdutoDTO dto) {
```

```
41     Produto produto = produtoService.create(dto);
```

```
42     // Cria o URI para o recurso criado
```

```
43     URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
```

```
44         .buildAndExpand(produto.getIdProduto()).toUri();
```

```
45     // Retorna a resposta com o status 201 Created e o local do recurso criado
```

```
46     return ResponseEntity.created(uri).build();
```

```
47 }
```

```
48 }
```

- Para testarmos nosso endpoint no Postman vamos copiar um Json de nosso Produto a partir de uma consulta que já implementamos em nosso projeto.
- Não precisaremos de todos os dados para inserir um novo grupo produto. Não são necessários: **id, datacadastro, descricaoGrupoProduto e valor estoque.**

## Criando Projeto Spring Boot – Create

```
{
  "idProduto": 1,
  "codigoBarra": "1111",
  "descricao": "Coca Cola",
  "saldoEstoque": 100.000,
  "valorUnitario": 3.500,
  "valorEstoque": 350.00,
  "dataCadastro": "23/10/2024",
  "grupoProduto": 2,
  "descricaoGrupoProduto": "Alimenticio",
  "status": 1
}
```



**A** POST **B** http://localhost:8080/produto

Params Authorization Headers **C** Body **D** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **E** JSON

```
1 {
2   "codigoBarra": "5555",
3   "descricao": "Fanta",
4   "saldoEstoque": 200.000,
5   "valorUnitario": 3.500,
6   "grupoProduto": 2,
7   "status": 1
8 }
```

**F**

Body Cookies Headers (5) Test Results 201 Created • 152 ms • 171 B

Key	Value
Location	http://localhost:8080/produto/5

- Configure seu Postman como requisição do tipo **Post** (A) informe nosso URI de acesso ao endpoint (B).
- Escolha a opção Body (C) e a opção RAW (D) e escolha JSON (E).
- Cole nosso JSON de Produto e altere com os novos dados a incluir e retire os atributos não necessários (F).
- Ai é só executar o teste (SEND)





# Spring Boot

## Endpoint Create – Produto

### Verificação de Integridade para código de barras

POST

http://localhost:8080/produto

Params Authorization Headers (8) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {  
2   "codigoBarra": "6666",  
3   "descricao": "Sprite",  
4   "saldoEstoque": 200.000,  
5   "valorUnitario": 3.000,  
6   "grupoProduto": 2,  
7   "status": 1  
8 }
```

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

Text



1

201 Created

## Criando Projeto Spring Boot – Erros

- Se tentarmos inserirmos novamente um novo produto com os mesmos dados teremos sucesso!
- **Mas seria correto que o código de barras de um produto ter duplicidade no banco de dados?**
- Temos que corrigir isso!!

```
@Entity 33 usages  jeffersonarpasserini +1 *
@Table(name = "produto")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_produto")
    private Long idProduto;

    @NotBlank @NotNull 4 usages
    @Column(unique = true)
    private String codigoBarra;

    @NotBlank @NotNull 7 usages
    private String descricao;

    @NotNull 5 usages
    @Digits(integer = 15, fraction = 3)
    private BigDecimal saldoEstoque;

    @NotNull 6 usages
    @Digits(integer = 15, fraction = 3)
    private BigDecimal valorUnitario;
```

- Na classe Produto, vamos inserir para o atributo codigoBarra a seguinte anotação:
- @Column(unique = true)
- Deste modo o atributo codigoBarra será tratado como uma **chave candidata** e não poderá se duplicado em nosso SGBD.

HTTP SuporteOS2024 / produto

Save Share

POST http://localhost:8080/produto Send

Params Auth Headers (8) Body Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "codigoBarra": "6666",
3   "descricao": "Sprite",
4   "saldoEstoque": 200.000,
5   "valorUnitario": 3.000,
6   "grupoProduto": 2,
7   "status": 1
8 }
```

Body 500 Internal Server Error • 53 ms • 14.4 KB

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2024-10-24T12:50:28.750+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "trace": "org.springframework.dao.DataIntegrityViolationException: could not execute
statement [Unique index or primary key violation: \"PUBLIC.CONSTRAINT_INDEX_1 ON
PUBLIC.PRODUTO(CODIGO_BARRA NULLS FIRST) VALUES ( /* 5 */ '6666' )\"]; SQL
statement:\ninsert into produto (codigo_barra,data_cadastro,descricao,
idgrupoproduto,saldo_estoque,status,valor_estoque,valor_unitario,id_produto)
values (?,?,?,?,?,?,?,?,?) [23505-224]] [insert into produto (codigo_barra,
data_cadastro,descricao,idgrupoproduto,saldo_estoque,status,valor_estoque,
valor_unitario,id_produto) values (?,?,?,?,?,?,?,?,?); SQL [insert into produto
(codigo_barra,data_cadastro,descricao,idgrupoproduto,saldo_estoque,status,
valor_estoque,valor_unitario,id_produto) values (?,?,?,?,?,?,?,?,?); constraint
[\"PUBLIC.CONSTRAINT INDEX 1 ON PUBLIC.PRODUTO(CODIGO BARRA NULLS FIRST) VALUES
```

Agora se tentarmos novamente **inserir um novo produto com os mesmos dados**

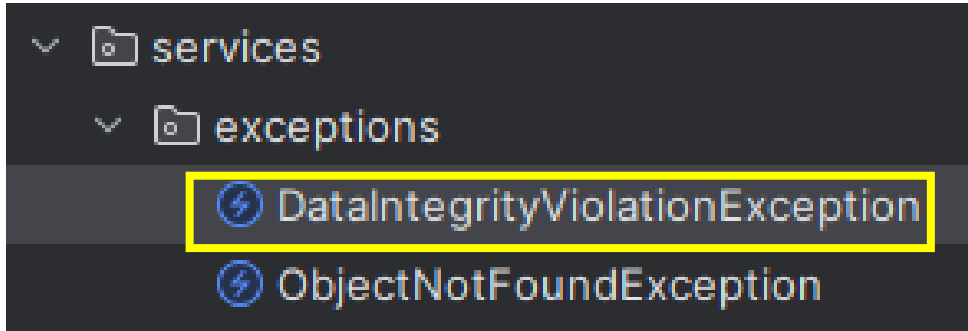
teremos um erro de integridade vindo do banco de dados, **pois o código de barras não pode ser duplicado** em nossa base.

Temos um erro do tipo: **Data Integrity Violation Exception.**

**Não é interessante deixarmos o erro ocorrer quando a informação chegar o banco de dados.**

**Devemos verificar essa situação para o código de barras antes de enviarmos para a Repository para gravar os dados.**

- Vamos preparar nosso tratamento de erro.
- No pacote **services** → **exceptions** crie a classe **DataIntegrityViolationException**.



```
1 package com.curso.services.exceptions;
2
3 public class DataIntegrityViolationException extends RuntimeException { no usag
4     private static final long serialVersionUID = 1L; no usages
5
6     public DataIntegrityViolationException(String message, Throwable cause){
7         super(message, cause);
8     }
9
10    public DataIntegrityViolationException(String message){ 1 usage  jeffersonarp
11        super(message);
12    }
13
14 }
```



## Criando Projeto Spring Boot – Erros

- Vamos preparar nosso tratamento de erro.
- No pacote **resources** → **exceptions** vamos alterar a classe **ResourceExceptionHandler**.
- E criar um método para o tratamento da exceção de **Data Integrity Violation** como indicado ao lado.

```
1 package com.curso.resources.exceptions;
2
3 > import ...
4
13
14 @ControllerAdvice  jeffersonarpasserini *
15 public class ResourceExceptionHandler {
16
17     @ExceptionHandler  jeffersonarpasserini
18     public ResponseEntity<StandardError> objectNotFoundException(ObjectNotFoundException ex, HttpServletRequest request)
19     {
20
21         StandardError error = new StandardError(System.currentTimeMillis(), HttpStatus.NOT_FOUND.value(),
22             error: "Object not found", ex.getMessage(), request.getRequestURI());
23
24         return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
25     }
26
27     @ExceptionHandler(MethodArgumentTypeMismatchException.class)  jeffersonarpasserini
28     public ResponseEntity<StandardError> handleMethodArgumentTypeMismatchException(
29         MethodArgumentTypeMismatchException ex, HttpServletRequest request)
30     {
31
32         StandardError error = new StandardError(System.currentTimeMillis(), HttpStatus.BAD_REQUEST.value(),
33             error: "Bad Request", ex.getMessage(), request.getRequestURI());
34
35         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
36     }
37
38     @ExceptionHandler(DataIntegrityViolationException.class) new *
39     public ResponseEntity<StandardError> dataIntegrityViolationException(DataIntegrityViolationException ex, HttpServletRequest request)
40     {
41
42         StandardError error = new StandardError(System.currentTimeMillis(), HttpStatus.BAD_REQUEST.value(), error: "Data Integrity Violation",
43             ex.getMessage(), request.getRequestURI());
44
45         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
46     }
47 }
```

### Repository

```
@Repository 4 usages jeffersonarpasserini  
public interface ProdutoRepository extends JpaRepository<Produto, Long> {  
    Optional<Produto> findByCodigoBarra(String codigoBarra); 1 usage jeffer  
}
```

**Precisamos verificar se já existe um produto previamente cadastrado com mesmo código de barras**

**Para isso já temos o recurso de busca por código de barras implementado, vamos utiliza-lo.**

- Implementando a validação de código de barras.
- Na classe **ProdutoService** vamos criar um método de validação pra produto – validaProduto(), digite conforme o exemplo.
- Também precisamos chamar o método de validação em nosso método Create() para que funcione.

```
1 package com.curso.services;
2
3 > import ...
4
5 @Service 4 usages 1 jeffersonarpasserini +1 *
6 public class ProdutoService {
7
8     @Autowired
9     private ProdutoRepository produtoRepo;
10
11     public List<ProdutoDTO> findAll(){ 1 usage 1 jeffersonarpasserini *
12         //retorna uma lista de ProdutoDTO
13         return produtoRepo.findAll().stream()
14             .map(obj -> new ProdutoDTO(obj)).collect(Collectors.toList());
15     }
16
17     public Produto findById(Long id){ 1 usage 1 jeffersonarpasserini
18         Optional<Produto> obj = produtoRepo.findById(id);
19         return obj.orElseThrow (() -> new ObjectNotFoundException("Produto não encontrado! Id: "+id));
20     }
21
22     public Produto findByCodigoBarra(String codigoBarra){ 1 usage 1 jeffersonarpasserini
23         Optional<Produto> obj = produtoRepo.findByCodigoBarra(codigoBarra);
24         return obj.orElseThrow(
25             () -> new ObjectNotFoundException("Produto não encontrado! CodigoBarra: "+codigoBarra)
26         );
27     }
28
29     @
30     public Produto create(ProdutoDTO dto){ new *
31         dto.setIdProduto(null);
32         validaProduto(dto);
33         Produto obj = new Produto(dto);
34         return produtoRepo.save(obj);
35     }
36
37     @
38     private void validaProduto(ProdutoDTO dto){ 1 usage new *
39         Optional<Produto> obj = produtoRepo.findByCodigoBarra(dto.getCodigoBarra());
40         if(obj.isPresent() && obj.get().getIdProduto() != dto.getIdProduto()){
41             throw new DataIntegrityViolationException("Código de barras já cadastrado!");
42         }
43     }
44
45 }
```



POST



http://localhost:8080/produto

Send



Params Auth Headers (8) Body ● Scripts Settings

Cookies

raw



JSON



Beautify

```
1 {
2   "codigoBarra": "6666",
3   "descricao": "Sprite",
4   "saldoEstoque": 200.000,
5   "valorUnitario": 3.000,
6   "grupoProduto": 2,
7   "status": 1
8 }
```

Body



400 Bad Request



26 ms



283 B



Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "timeStamp": 1729776928903,
3   "status": 400,
4   "error": "Data Integrity Violation",
5   "message": "Código de barras já cadastrado!",
6   "path": "/produto"
7 }
```

## Criando Projeto Spring Boot – Erros

- Agora se tentarmos inserirmos duas vezes o mesmo registro com o mesmo código de barras o sistema irá lançar a exceção **Data Integrity Violation** para o código de barras.
- É sempre interessante que façamos as validações antes de ocorrer o erro no banco de dados.



POST

http://localhost:8080/produto

Send

Params Auth Headers (8) Body Scripts Settings

Cookies

raw

JSON

Beautify

```
1 {
2   "codigoBarra": "5555",
3   "descricao": "Fata uva",
4   "saldoEstoque": 200.000,
5   "valorUnitario": 2.000,
6   "grupoProduto": 3,
7   "status": 1
8 }
```

Body

500 Internal Server Error

38 ms • 13.99 KB • e.g. ...

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "timestamp": "2024-10-24T13:37:49.375+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "trace": "org.springframework.dao.DataIntegrityViolationException: could not execute
statement [Referential integrity constraint violation:
\"FKDUPRCMOX214HSCOT5CPU30D316: PUBLIC.PRODUTO FOREIGN KEY(IDGRUPOPRODUTO)
REFERENCES PUBLIC.GRUPOPRODUTO(ID) (3)\"; SQL statement:\ninsert into produto
(codigo_barra,data_cadastro,descricao,idgrupoproduto,saldo_estoque,status,
valor_estoque,valor_unitario,id_produto) values (?,?,?,?,?,?,?,?) [23506-224]]
[insert into produto (codigo_barra,data_cadastro,descricao,idgrupoproduto,
saldo_estoque,status,valor_estoque,valor_unitario,id_produto) values
(?,?,?,?,?,?,?,?); SQL [insert into produto (codigo_barra,data_cadastro,
descricao,idgrupoproduto,saldo_estoque,status,valor_estoque,valor_unitario,
```

## Criando Projeto Spring Boot – Erros

- Observe agora o erro desta requisição, temos novamente o erro de **Data Integrity Violation**
- Só que agora estamos tentando inserir um produto com um grupo produto que não está previamente cadastrado!**
- Utilizando a mesma lógica do exemplo anterior, vamos também efetuar essa validação!**



- **Altere** novamente o método **validaProduto()** na classe **ProdutoService** e insira a verificação para o grupo de produto.
- Faça conforme o exemplo!

```
private void validaProduto(ProdutoDTO dto){ 1 usage new *
    Optional<Produto> obj = produtoRepo.findByCodigoBarra(dto.getCodigoBarra());
    if(obj.isPresent() && obj.get().getIdProduto() != dto.getIdProduto()){
        throw new DataIntegrityViolationException("Código de barras já cadastrado!");
    }

    Optional<GrupoProduto> grupoProduto = grupoProdutoRepo.findById(dto.getGrupoProduto());
    if(!grupoProduto.isPresent()){
        throw new DataIntegrityViolationException("Grupo de Produto - " + dto.getGrupoProduto() + " não está cadastrado!");
    }
}
```

- **Agora temos nosso erro devidamente tratado!**

HTTP SuporteOS2024 / produto Save Share

**POST** http://localhost:8080/produto Send

Params Auth Headers (8) **Body** Scripts Settings Cookies

raw **JSON** Beautify

```
1 {
2   "codigoBarra": "5555",
3   "descricao": "Fata uva",
4   "saldoEstoque": 200.000,
5   "valorUnitario": 3.000,
6   "grupoProduto": 3,
7   "status": 1
8 }
```

Body 400 Bad Request • 19 ms • 293 B • 🌐 📄 ⋮

Pretty Raw Preview Visualize **JSON** 🔍

```
1 {
2   "timeStamp": 1729777702605,
3   "status": 400,
4   "error": "Data Integrity Violation",
5   "message": "Grupo de Produto - 3 não está cadastrado!",
6   "path": "/produto"
7 }
```



# Spring Boot

## Endpoint Create – Annotations

### Validação de Dados – não nulo

## Criando Projeto Spring Boot – Validação

```
public class ProdutoDTO { 15 usages  ↳ jeffersonarpasserini +1 *

    private Long idProduto; 3 usages

    @NotBlank(message = "O campo código de barras não pode estar vazio") 3
    @NotNull(message = "O campo código de barras não pode ser nulo")
    private String codigoBarra;

    @NotNull(message = "O campo descrição 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") 2 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
```

- Como sabemos nossas classes DTO serão o canal de comunicação da interface com o backend.
- Assim definimos uma série de validações nesta classe, como pode-se verificar no exemplo ao lado.
- **Essas validações ainda não estão sendo realizadas, vamos corrigir isso!**
- **As validações devem ser feitas no endpoint, na entrada do dado no backend.**
- **Assim vamos tratar o endpoint Create() de cada uma das classes.**

## Criando Projeto Spring Boot – Validação

- A entrada dos dados ocorre no endpoint create() na ProdutoResource.
- Esse endpoint recebe o ProdutoDTO que está anotado com as validações.
- **Devemos então anotar esse endpoint para que faça as validações de acordo com as anotações do DTO.**
- **Inclua @Valid na declaração de parâmetros do método Create().**

```
@RestController
@jeffersonarpasserini +1 *
@RequestMapping(value = "/produto")
public class ProdutoResource {

    @Autowired
    private ProdutoService produtoService;

    @GetMapping //exemplo http://localhost:8080/produto
    public ResponseEntity<List<ProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(produtoService.findAll());
    }

    @GetMapping(value =("/{id}") //exemplo http://localhost:8080/produto/1
    public ResponseEntity<ProdutoDTO> findById(@PathVariable Long id){
        Produto obj = this.produtoService.findById(id);
        return ResponseEntity.ok().body(new ProdutoDTO(obj));
    }

    @GetMapping(value = ("/codigo barra/{codigoBarra}") //exemplo http://localhost:8
    public ResponseEntity<ProdutoDTO> findById(@PathVariable String codigoBarra){
        Produto obj = this.produtoService.findbyCodigoBarra(codigoBarra);
        return ResponseEntity.ok().body(new ProdutoDTO(obj));
    }

    @PostMapping
    public ResponseEntity<ProdutoDTO> create(@Valid @RequestBody ProdutoDTO dto) {
        Produto produto = produtoService.create(dto);
        // Cria o URI para o recurso criado
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
            .buildAndExpand(produto.getIdProduto()).toUri();
        // Retorna a resposta com o status 201 Created e o local do recurso criado
        return ResponseEntity.created(uri).build();
    }
}
```



- **Faça o mesmo com o endpoint `Create()` na classe `GrupoProdutoResource`.**

```
@RestController  @jeffersonarpasserini *
@RequestMapping(value = "/grupoproduto")
public class GrupoProdutoResource {

    @Autowired
    private GrupoProdutoService grupoProdutoService;

    @GetMapping //exemplo http://localhost:8080/produto  @jeffersonarpasserini
    public ResponseEntity<List<GrupoProdutoDTO>> findAll(){
        return ResponseEntity.ok().body(grupoProdutoService.findAll());
    }

    @GetMapping(value =("/{id}") //exemplo http://localhost:8080/grupoproduto/1  @jeffersonar
    public ResponseEntity<GrupoProdutoDTO> findById(@PathVariable Integer id){
        GrupoProduto obj = this.grupoProdutoService.findById(id);
        return ResponseEntity.ok().body(new GrupoProdutoDTO(obj));
    }

    @PostMapping new *
    public ResponseEntity<GrupoProdutoDTO> create(@Valid @RequestBody GrupoProdutoDTO dto) {
        GrupoProduto grupoProduto = grupoProdutoService.create(dto);
        // Cria o URI para o recurso criado
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
            .buildAndExpand(grupoProduto.getId()).toUri();
        // Retorna a resposta com o status 201 Created e o local do recurso criado
        return ResponseEntity.created(uri).build();
    }
}
```



```
public class ProdutoDTO { 15 usages  jeffersonarpasserini +1 *  
  
    private Long idProduto; 3 usages  
  
    @NotBlank(message = "O campo código de barras não pode estar vazio") 3  
    @NotNull(message = "O campo código de barras não pode ser nulo")  
    private String codigoBarra;  
  
    @NotNull(message = "O campo descrição 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") 2 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
```

- Antes de continuarmos, vamos corrigir um detalhe em nosso ProdutoDTO.
- O atributo valorEstoque é um atributo calculado, esse cálculo é realizado na classe Produto, assim não precisamos recebê-lo via ProdutoDTO.
- Remova o atributo valorEstoque da classe EstoqueDTO e corrija o construtor e os Getters e Setters.

## Criando Projeto Spring Boot – Validação

- Após colocarmos o **@Valid** em nossos endpoints **Create()** se fizermos uma requisição que fere as anotações no DTO o Spring gerará uma exceção.
- Neste caso estamos passando **valor nulo para a descrição do produto**, o que fere a anotação deste campo, gerando uma exceção do tipo: **MethodArgumentNotValidException**.
- Vamos tratar essas ocorrências!

POST http://localhost:8080/produto

Params Auth Headers (8) Body Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "codigoBarra": "5555",
3   "descricao": null,
4   "saldoEstoque": 200.000,
5   "valorUnitario": 3.000,
6   "grupoProduto": 2,
7   "status": 1
8 }
```

Body 400 Bad Request • 45 ms • 6.9 KB

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2024-10-24T15:42:53.340+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "trace": "org.springframework.web.bind.MethodArgumentNotValidException: Validation
failed for argument [0] in public org.springframework.http.ResponseEntity<com.
curso.domains.dtos.ProdutoDTO> com.curso.resources.ProdutoResource.create(com.
curso.domains.dtos.ProdutoDTO) with 2 errors: [Field error in object 'produtoDTO'
on field 'descricao': rejected value [null]; codes [NotNull.produtoDTO.descricao,
NotNull.descricao,NotNull.java.lang.String,NotNull]; arguments [org.
springframework.context.support.DefaultMessageSourceResolvable: codes [produtoDTO.
descricao,descricao]; arguments []; default message [descricao]]; default message
```

- Para tratarmos a exceção crie dentro de **resources** → **exceptions** as classes **FieldMessage** e **ValidationError**



- **Codifique a classe FieldMessage conforme o exemplo.**
- **Esta classe irá gerar objetos que armazenam os erros de cada atributo de nossas classes DTO.**

```
1 package com.curso.resources.exceptions;
2
3 public class FieldMessage { 3 usages jeffersonarpasserini
4
5     private String fieldName; 3 usages
6     private String message; 3 usages
7
8     > public FieldMessage() { super(); }
9
10
11
12     < public FieldMessage(String fieldName, String message) { 1
13         this.fieldName = fieldName;
14         this.message = message;
15     }
16
17     < public String getFieldName() { no usages jeffersonarpasserini
18         return fieldName;
19     }
20
21     < public void setFieldName(String fieldName) { no usages je
22         this.fieldName = fieldName;
23     }
24
25     < public String getMessage() { no usages jeffersonarpasserini
26         return message;
27     }
28
29     < public void setMessage(String message) { no usages jeffersc
30         this.message = message;
31     }
32
33 }
```

```
1 package com.curso.resources.exceptions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class ValidationError extends StandardError { no usages new *
7
8     private List<FieldMessage> errors = new ArrayList<>(); 2 usages
9
10    > public ValidationError() { super(); }
11
12
13
14    public ValidationError(Long timeStamp, Integer status, String error, String message, String path) {
15        | super(timeStamp, status, error, message, path);
16    }
17
18    public List<FieldMessage> getErrors() { new *
19        | return errors;
20    }
21
22    public void addErrors(String fieldName, String message) { no usages new *
23        | this.errors.add(new FieldMessage(fieldName, message));
24    }
25 }
```

- **Agora implemente a classe `ValidationError` conforme o exemplo.**
- **Esta classe estende a classe `StandardError` e armazenará a lista de `FieldMessages` dos atributos que geraram exceções na validação.**

- Quando ocorre a validação dos atributos da DTO ocorre um erro do tipo:  
**MethodArgumentNotValidException.**
- Na classe **ResourceExceptionHandler** no pacote **resources→exceptions** vamos criar um método para tratar esse tipo de exceção, o método está definido abaixo.
- Os erros deste tipo informam todas as ocorrências de uma única vez, por esse motivo temos uma lista de erros.**

```
@ExceptionHandler(MethodArgumentNotValidException.class) new *
public ResponseEntity<StandardError> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex, HttpServletRequest request){

    ValidationError errors = new ValidationError(System.currentTimeMillis(),HttpStatus.BAD_REQUEST.value(), error: "Data Validation Error",
        message: "Field Validation Error",request.getRequestURI());

    for(FieldError x : ex.getBindingResult().getFieldErrors()){
        errors.addErrors(x.getField(), x.getDefaultMessage());
    }

    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
}
```



## Criando Projeto Spring Boot – Validação

- No teste ao lado podemos observar que o atributo descrição nulo o que gerou as ocorrências (1) e (2).
- No caso do atributo saldoEstoque ele está sendo informado como NULL então gerou o evento de exceção o de campo nulo (3).

POST

http://localhost:8080/produto

Params

Authorization

Headers (8)

Body

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 {
2   "codigoBarra": "5555",
3   "descricao": null,
4   "saldoEstoque": null,
5   "valorUnitario": 3.000,
6   "grupoProduto": 2,
7   "status": 1
8 }
```

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```
4   "error": "Data Validation Error",
5   "message": "Field Validation Error",
6   "path": "/produto",
7   "errors": [
8     {
9       "fieldName": "descricao",
10      "message": "O campo descrição não pode ser nulo"
11    },
12    {
13      "fieldName": "descricao",
14      "message": "O campo descrição não pode estar vazio"
15    },
16    {
17      "fieldName": "saldoEstoque",
18      "message": "O campo saldoEstoque não pode ser nulo"
19    }
20  ]
21 }
```



# Spring Boot Endpoint Update - GrupoProducto

@Service 2 usages jeffersonarpasserini +1\*

```
public class GrupoProdutoService {}
```

@Autowired

```
private GrupoProdutoRepository grupoProdutoRepo;
```

```
public List<GrupoProdutoDTO> findAll(){ 1 usage jeffersonarpasserini
//retorna uma lista de ProdutoDTO
return grupoProdutoRepo.findAll().stream() Stream<GrupoProduto>
    .map(obj -> new GrupoProdutoDTO(obj)) Stream<GrupoProduto>
    .collect(Collectors.toList());
}
```

```
public GrupoProduto findById(int id){ 2 usages jeffersonarpasserini
Optional<GrupoProduto> obj = grupoProdutoRepo.findById(id);
return obj.orElse( other: null);
}
```

```
public GrupoProduto create(GrupoProdutoDTO dto){ new *
dto.setId(null);
GrupoProduto obj = new GrupoProduto(dto);
return grupoProdutoRepo.save(obj);
}
```

```
public GrupoProduto update(Integer id, GrupoProdutoDTO objDto){
    objDto.setId(id);
    GrupoProduto oldObj = findById(id);
    oldObj = new GrupoProduto(objDto);
    return grupoProdutoRepo.save(oldObj);
}
```

- Vamos criar na Service (**GrupoProdutoService**) o método responsável por receber os dados (DTO) da camada **Resources** (Controllers).
- Esse método deverá converter o padrão de dados de **DTO para Domain** e solicitar a camada **Repository** que faça a gravação através do método **SAVE()**.
- **Antes ele irá verificar se o grupo produto a ser alterado realmente existe, se sim realiza a alteração dos dados.**

- Em **GrupoProdutoResource** vamos criar o **endpoint** para receber a requisição de alteração do tipo (**PutMapping**).
- Adicione o método descrito abaixo a sua classe GrupoProdutoResource.

```
@PutMapping(value = "{id}") new *
public ResponseEntity<GrupoProdutoDTO> update(@PathVariable Integer id, @Valid @RequestBody GrupoProdutoDTO objDto){
    GrupoProduto Obj = grupoProdutoService.update(id, objDto);
    return ResponseEntity.ok().body(new GrupoProdutoDTO(Obj));
}
```

HTTP SuporteOS2024 / grupoproduto

PUT http://localhost:8080/grupoproduto/1

Params Authorization Headers (8) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON

```
1 {
2   "id": 1,
3   "descricao": "Produtos de Limpeza",
4   "status": 1
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "descricao": "Produtos de Limpeza",
4   "status": 1
5 }
```

- Os dados originais são:

```
{
  "id": 1,
  "descricao": "Limpeza",
  "status": 1
}
```

- Os dados da descrição foi alterado de "Limpeza" para "Produtos de Limpeza".





# Spring Boot Endpoint Update - Produto



## Criando Projeto Spring Boot – Update

```
private void validaProduto(ProdutoDTO dto){ 2 usages new *
    Optional<Produto> obj = produtoRepo.findByCodigoBarra(dto.getCodigoBarra());
    if(obj.isPresent() && obj.get().getIdProduto() != dto.getIdProduto()){
        throw new DataIntegrityViolationException("Código de barras já cadastrado!");
    }

    Optional<GrupoProduto> grupoProduto = grupoProdutoRepo.findById(dto.getGrupoProduto());
    if(!grupoProduto.isPresent()){
        throw new DataIntegrityViolationException("Grupo de Produto - " + dto.getGrupoProduto() + " ");
    }
}
```

```
public Produto update(Long id, ProdutoDTO objDto){ new *
    objDto.setIdProduto(id);
    Produto oldObj = findById(id);
    validaProduto(objDto);
    oldObj = new Produto(objDto);
    return produtoRepo.save(oldObj);
}
```

- Vamos criar na Service (**ProdutoService**) o método responsável por receber os dados (DTO) da camada **Resources** (Controllers).
- Esse método deverá converter o padrão de dados de **DTO para Domain** e solicitar a camada **Repository** que faça a gravação através do método **SAVE()**.
- **Antes ele irá verificar se o produto a ser alterado realmente existe, se sim realiza a alteração dos dados.**

- Em **ProdutoResource** vamos criar o **endpoint** para receber a requisição de alteração do tipo (**PutMapping**).
- Adicione o método descrito abaixo a sua classe ProdutoResource.

```
@PostMapping(new *  
public ResponseEntity<ProdutoDTO> create(@Valid @RequestBody ProdutoDTO dto) {  
    Produto produto = produtoService.create(dto);  
    // Cria o URI para o recurso criado  
    URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")  
        .buildAndExpand(produto.getIdProduto()).toUri();  
    // Retorna a resposta com o status 201 Created e o local do recurso criado  
    return ResponseEntity.created(uri).build();  
}
```

```
@PutMapping(value = "{id}") new *  
public ResponseEntity<ProdutoDTO> update(@PathVariable Long id, @Valid @RequestBody ProdutoDTO objDto){  
    Produto Obj = produtoService.update(id, objDto);  
    return ResponseEntity.ok().body(new ProdutoDTO(Obj));  
}
```

PUT ▼ http://localhost:8080/produto/1

Params Authorization Headers (8) **Body** ● Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ Graph

```
1 {
2   "idProduto": 1,
3   "codigoBarra": "1111",
4   "descricao": "Coca Cola Zero",
5   "saldoEstoque": 100.000,
6   "valorUnitario": 3.500,
7   "dataCadastro": "24/10/2024",
8   "grupoProduto": 2,
9   "descricaoGrupoProduto": "Alimenticio",
10  "status": 1
11 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1 {
2   "idProduto": 1,
3   "codigoBarra": "1111",
4   "descricao": "Coca Cola Zero",
5   "saldoEstoque": 100.000,
6   "valorUnitario": 3.500,
7   "dataCadastro": "24/10/2024",
8   "grupoProduto": 2,
9   "descricaoGrupoProduto": "Alimenticio",
10  "status": 1
11 }
```

- Os dados originais são:

```
{
  "idProduto": 1,
  "codigoBarra": "1111",
  "descricao": "Coca Cola",
  "saldoEstoque": 100.000,
  "valorUnitario": 3.500,
  "dataCadastro": "24/10/2024",
  "grupoProduto": 2,
  "descricaoGrupoProduto": "Alimenticio",
  "status": 1
}
```

- Os dados da descrição foi alterado de "Coca Cola" para "Coca Cola Zero".



# Spring Boot Endpoint Delete - GrupoProducto

```
public GrupoProduto update(Integer id, GrupoProdutoDTO objDto){ new *
    objDto.setId(id);
    GrupoProduto oldObj = findById(id);
    oldObj = new GrupoProduto(objDto);
    return grupoProdutoRepo.save(oldObj);
}
```

```
public void delete(Integer id){ 1 usage new *
    GrupoProduto obj = findById(id);
    if (obj.getProdutos().size()>0){
        throw new DataIntegrityViolationException("Grupo de produto não pode ser deletado pois possui produtos vinculados!");
    }
    grupoProdutoRepo.deleteById(id);
}
```

- Na camada Services crie o método para realizar a operação delete para os grupos de produto (**GrupoProdutoService**).
- **Se houver produtos para o grupo de produtos não poderá ser apagado.**

```
@PostMapping new *
public ResponseEntity<GrupoProdutoDTO> create(@Valid @RequestBody GrupoProdutoDTO dto) {
    GrupoProduto grupoProduto = grupoProdutoService.create(dto);
    // Cria o URI para o recurso criado
    URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
        .buildAndExpand(grupoProduto.getId()).toUri();
    // Retorna a resposta com o status 201 Created e o local do recurso criado
    return ResponseEntity.created(uri).build();
}

@PutMapping(value = "{id}") new *
public ResponseEntity<GrupoProdutoDTO> update(@PathVariable Integer id, @Valid @RequestBody GrupoProdutoDTO objDto){
    GrupoProduto Obj = grupoProdutoService.update(id, objDto);
    return ResponseEntity.ok().body(new GrupoProdutoDTO(Obj));
}

@DeleteMapping(value = "{id}") new *
public ResponseEntity<GrupoProdutoDTO> delete(@PathVariable Integer id){
    grupoProdutoService.delete(id);
    return ResponseEntity.noContent().build();
}
```

- Na camada resources crie o endpoint para atender a solicitação de delete para os grupos de produtos (**GrupoProdutoService**). **Faça os testes do endpoint no Postman.**





# Spring Boot Endpoint Delete - Produto

```
if(!grupoProduto.isPresent()){  
    throw new DataIntegrityViolationException("Grupo de Produto - " + dto.getGrupoProduto() + " não está cadastrado!");  
}  
}  
  
public Produto update(Long id, ProdutoDTO objDto){ new *  
    objDto.setIdProduto(id);  
    Produto oldObj = findById(id);  
    validaProduto(objDto);  
    oldObj = new Produto(objDto);  
    return produtoRepo.save(oldObj);  
}  
  
public void delete(Long id){ new *  
    Produto obj = findById(id);  
    produtoRepo.deleteById(id);  
}  
}
```

- Na camada Services crie o método para realizar a operação delete para os produtos (**ProdutoService**).

```
@PutMapping(value = "{id}") new *
public ResponseEntity<ProdutoDTO> update(@PathVariable Long id, @Valid @RequestBody ProdutoDTO objDto){
    Produto Obj = produtoService.update(id, objDto);
    return ResponseEntity.ok().body(new ProdutoDTO(Obj));
}
```

```
@DeleteMapping(value = "{id}") new *
public ResponseEntity<GrupoProdutoDTO> delete(@PathVariable Long id){
    produtoService.delete(id);
    return ResponseEntity.noContent().build();
}
```

- Na camada resources crie o endpoint para atender à solicitação de delete para os produtos (**ProdutoService**). **Faça os testes do endpoint no Postman.**