



FIAP



Domain Driven Design using Java





AGENDA




1

Persistência de Dados com Spring Data JPA utilizando Oracle

2

Validações com Bean Validation



Spring Data JPA com Oracle


Configuração do Spring Data JPA

- Dependência Maven no pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>19.8.0.0</version>
</dependency>
```

Configuração do Spring Data JPA

- Configuração do banco de dados no application.properties:

```
1  spring.application.name=estoque
2   spring.datasource.url=jdbc:oracle:thin:@oracle.fiap.com.br:1521:ORCL
3  spring.datasource.username=USUARIO
4  spring.datasource.password=SENHA
5  spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
6  spring.jpa.hibernate.ddl-auto=update
7  spring.h2.console.enabled=true
```

Criando entidade

Definição da Classe Entidade

```
@Entity
@Table(name = "PRODUTO")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_PRODUTO")
    private Long id;

    @Column(name = "NOME", nullable = false, length = 100)
    private String nome;

    @Column(name = "PRECO", nullable = false)
    private BigDecimal preco;

    @Column(name = "QUANTIDADE", nullable = false)
    private Integer quantidade;

    @Column(name = "DATA_CRIACAO", updatable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date dataCriacao;

    // Getters e Setters
}
```


Bean Validation

Validações com Bean Validation

- O que é Bean Validation?

- É uma especificação Java para validação de dados, utilizada em conjunto com frameworks como Spring.
- Reduz erros e melhora a confiabilidade do sistema ao garantir que os dados respeitam regras pré-definidas antes de serem armazenados no banco.

- Por que usar Bean Validation?

- Evita que dados inválidos sejam persistidos no banco de dados.
- Reduz a necessidade de validações manuais dentro do código.
- Melhora a integridade e segurança dos dados no sistema.

Validações com Bean Validation

```
@Entity
@Table(name = "PRODUTO")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_PRODUTO")
    private Long id;

    @NotBlank(message = "O nome do produto é obrigatório")
    @Size(max = 100, message = "O nome deve ter no máximo 100 caracteres")
    @Column(name = "NOME", nullable = false, length = 100)
    private String nome;

    @NotNull(message = "O preço deve ser informado")
    @DecimalMin(value = "0.1", message = "O preço deve ser maior que zero")
    @Column(name = "PRECO", nullable = false)
    private BigDecimal preco;

    @NotNull(message = "A quantidade deve ser informada")
    @Min(value = 1, message = "A quantidade deve ser ao menos 1 unidade")
    @Column(name = "QUANTIDADE", nullable = false)
    private Integer quantidade;

    @PastOrPresent(message = "A data de criação deve ser válida")
    @Column(name = "DATA_CRIACAO", updatable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date dataCriacao;
}
```

Validações com Bean Validation

- O Spring Boot, ao utilizar **Spring MVC**, automaticamente valida os objetos recebidos pelo controlador. Para garantir que os dados estão corretos antes da gravação no banco de dados, podemos usar o *@Valid* no endpoint:

```
@PostMapping
public Produto criar(@RequestBody @Valid Produto produto) {
    return produtoRepository.save(produto);
}
```

Exemplo de Erros de Validação

- Se um usuário tentar cadastrar um produto sem nome ou com preço negativo, a API retornará uma resposta com mensagens de erro, garantindo que apenas dados corretos sejam persistidos.

Exemplo de resposta em JSON:

```
{
  "timestamp": "2025-05-11T18:45:00",
  "status": 400,
  "errors": [
    "O nome do produto é obrigatório",
    "O preço deve ser maior que zero"
  ]
}
```

Criando repositório

Definição da Interface Repository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProdutoRepository extends JpaRepository<Produto, Long> {
    List<Produto> findByNomeContaining(String nome);
}
```

- *JpaRepository<Produto, Long>* → Fornece métodos CRUD automaticamente.
- *findByNomeContaining* → Busca produtos pelo nome com filtros dinâmicos.

Criando o Serviço

Definição da Classe Service

```
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ProdutoService {

    private final ProdutoRepository produtoRepository;

    public ProdutoService(ProdutoRepository produtoRepository) {
        this.produtoRepository = produtoRepository;
    }

    public List<Produto> listarTodos() {
        return produtoRepository.findAll();
    }

    public Produto salvar(Produto produto) {
        return produtoRepository.save(produto);
    }

    public void deletar(Long id) {
        produtoRepository.deleteById(id);
    }
}
```

- **@Service** → Define a classe como um serviço de regras de negócio.
- Utiliza *produtoRepository* para acessar o banco de dados.

Criando o Controller

Definição da Classe Service

```
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/produtos")
public class ProdutoController {

    private final ProdutoService produtoService;

    public ProdutoController(ProdutoService produtoService) {
        this.produtoService = produtoService;
    }

    @GetMapping
    public List<Produto> listarTodos() {
        return produtoService.listarTodos();
    }

    @PostMapping
    public Produto salvar(@RequestBody Produto produto) {
        return produtoService.salvar(produto);
    }

    @DeleteMapping("/{id}")
    public void deletar(@PathVariable Long id) {
        produtoService.deletar(id);
    }
}
```

Referências

- **Documentação Oficial:** <https://spring.io/projects/spring-data-jpa>.
- Livro: *Spring in Action* - Craig Walls.



FIAP

