

BACK-END

Prof. Bruno Kurzawe



Persistência de Dados

Persistência de dados é o processo de armazenar e manter informações (dados) de forma duradoura, de modo que possam ser recuperadas e utilizadas posteriormente. Esse conceito é fundamental em ciência da computação e em desenvolvimento de software, pois a maioria das aplicações precisa armazenar e recuperar dados em algum momento.

Quando falamos de persistir, podemos persistir em banco de dados relacionais e não relacionais.

Exemplos de Banco de dados Relacionais



Exemplos de Banco de dados Não Relacionais



Primeira coisa é importar o driver do banco e isso já fizemos quando criamos nossa aplicação.

O Hibernate é um framework de mapeamento objeto-relacional (ORM) de código aberto para a linguagem Java. Ele fornece uma maneira eficiente e simplificada de mapear objetos Java para tabelas de banco de dados relacionais, permitindo que os desenvolvedores trabalhem com objetos em vez de lidar diretamente com SQL. O Hibernate lida com as complexidades da persistência de dados, como a criação, atualização, exclusão e recuperação de objetos em um banco de dados.

A Java Persistence API (JPA) é uma especificação padrão da Java EE (Enterprise Edition) para mapeamento objeto-relacional em aplicações Java. Ela define uma interface comum para trabalhar com ORM em Java e é implementada por diversos frameworks ORM, sendo o Hibernate uma das implementações mais conhecidas. A JPA permite que os desenvolvedores escrevam código que seja independente do provedor de ORM, o que facilita a troca de implementações.

Mapeando entidades JPA

@Entity

A anotação `@Entity` é parte da Java Persistence API (JPA) e é usada para marcar uma classe Java como uma entidade, indicando que essa classe deve ser mapeada para uma tabela em um banco de dados relacional. Uma entidade representa um objeto que será persistido no banco de dados e possui uma correspondência direta com uma tabela no esquema do banco de dados.

@Entity

```
public class Produto extends ItemVendavel {  
    private String nome;  
    private Double precoCompra;  
    private LocalDate dataValidade;  
    private LocalDate dataPrazo;  
    private Status status;  
  
    public Produto() {  
    }  
}
```

Esse erro acontece por que não mapeamos o ID ainda e obrigatoriamente temos que ter um ID definido.

Onde está o ID de produto?

```
package com.satc.satcloja.model;

public class EntityId {

    private Long id;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

}
```

```
package com.satc.satcloja.model;

import javax.persistence.*;

@MappedSuperclass
public class EntityId {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "id", nullable = false)
    private Long id;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
}
```


@MappedSuperclass

A anotação `@MappedSuperclass` é usada no contexto de mapeamento objeto-relacional (ORM) com frameworks como o JPA (Java Persistence API), que inclui o Hibernate. Essa anotação é usada para indicar que uma classe é uma superclasse da qual outras classes herdam propriedades de mapeamento. No entanto, a classe anotada com `@MappedSuperclass` não será mapeada para uma tabela no banco de dados por si só.

@Id

A anotação `@Id` é usada no contexto da Java Persistence API (JPA) para marcar um campo ou método getter como a chave primária de uma entidade mapeada. A chave primária é um atributo que identifica exclusivamente uma entrada na tabela do banco de dados e é usado para recuperar, atualizar ou excluir registros de forma eficiente.

@GeneratedValue

A anotação `@GeneratedValue` é usada no contexto da Java Persistence API (JPA) para especificar como os valores da chave primária (ou outros valores gerados) devem ser gerados automaticamente pelo sistema de banco de dados. Essa anotação é frequentemente usada em conjunto com a anotação `@Id` para indicar que um campo representa uma chave primária e que seu valor será gerado automaticamente.

@Column

A anotação `@Column` é usada no contexto da Java Persistence API (JPA) para mapear um atributo de uma classe Java para uma coluna em uma tabela de banco de dados. Essa anotação é usada principalmente para personalizar as propriedades de mapeamento da coluna, como nome da coluna, tipo de dado, tamanho e restrições.



`@MappedSuperclass`

```
public class ItemVendavel extends EntityId {
```

```
    @Column(name = "descricao", nullable = false)
```

```
    private String descricao;
```

```
    @Column(name = "valor_unitario", nullable = false)
```

```
    private Double valorUnitario;
```

```
    @Column(name = "estocavel", nullable = true)
```

```
    private Boolean estocavel;
```

```
@Entity
public class Produto extends ItemVendavel {

    @Column(name = "nome", length = 100, nullable = false)
    private String nome;
    @Column(name = "preco_compra", nullable = false)
    private Double precoCompra;
    @Column(name = "dt_validade", nullable = false)
    private LocalDate dataValidade;
    @Column(name = "dt_prazo", nullable = false)
    private LocalDate dataPrazo;
    @Enumerated(EnumType.STRING)
    @Column(name = "status")
    private Status status;
```

Rodando a aplicação



```
2023-08-28 18:05:20.243 INFO 315884 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.15.Final
2023-08-28 18:05:20.455 INFO 315884 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2023-08-28 18:05:20.573 INFO 315884 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-08-28 18:05:22.196 INFO 315884 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-08-28 18:05:22.210 INFO 315884 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.SQLServer2012Dialect
Hibernate: create table produto (id bigint not null, descricao varchar(255) not null, estocavel bit, valor_unitario double precision not null, dt_prazo date not null, dt_validade date not null, nome varchar(100) not null, preco_compra double precision not null, status varchar(255), primary key (id))
2023-08-28 18:05:23.969 INFO 315884 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-08-28 18:05:23.982 INFO 315884 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-08-28 18:05:24.064 WARN 315884 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2023-08-28 18:05:24.536 INFO 315884 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-08-28 18:05:24.553 INFO 315884 --- [main] com.satc.satcloja.SatclojaApplication : Started SatclojaApplication in 6.857 seconds (JVM running for 7.365s)
```

Mapeando a classe serviço

```
public class Servico extends ItemVendavel {  
    private Double quantidadeHoras;  
  
    public Servico(String descricao, Double quantidadeHoras, Double valor) {  
        super.setDescricao(descricao);  
        this.quantidadeHoras = quantidadeHoras;  
        super.setValorUnitario(valor);  
    }  
  
    public Double getQuantidadeHoras() { return quantidadeHoras; }  
  
    public void setQuantidadeHoras(Double quantidadeHoras) { this.quantidadeHoras = quantidadeHoras; }
```

Mapeando a classe serviço

```
package com.satc.satcloja.model;

import javax.persistence.Entity;

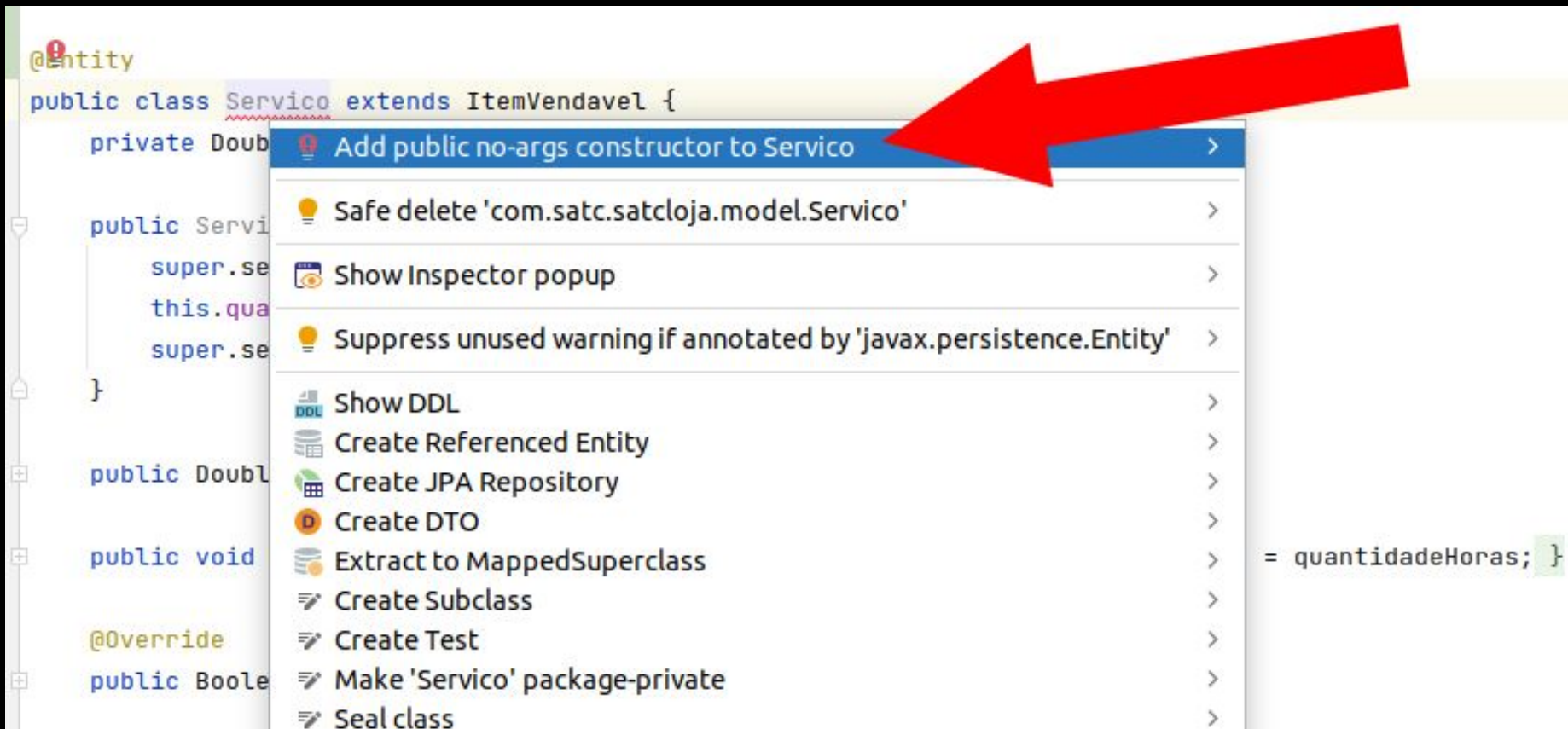
@Entity
public class Servico extends Vendavel {
    private Double quantidadeHoras;

    public Servico(String descricao, Double quantidadeHoras, Double valor) {
        super.setDescricao(descricao);
        this.quantidadeHoras = quantidadeHoras;
        super.setValorUnitario(valor);
    }

    public Double getQuantidadeHoras() { return quantidadeHoras; }

    public void setQuantidadeHoras(Double quantidadeHoras) { this.quantidadeHoras = quantidadeHoras; }
```

Mapeando a classe serviço



Mapeando a classe serviço

```
@Entity
public class Servico extends ItemVendavel {
    private Double quantidadeHoras;


    public Servico() {
    }

    public Servico(String descricao, Double quantidadeHoras, Double valor) {
        super.setDescricao(descricao);
        this.quantidadeHoras = quantidadeHoras;
        super.setValorUnitario(valor);
    }

    public Double getQuantidadeHoras() { return quantidadeHoras; }

    public void setQuantidadeHoras(Double quantidadeHoras) { this.quantidadeHoras = quantidadeHoras; }

    @Override
    public Boolean getEstocavel() { return false; }
```



Mapeando a classe serviço

```
@Entity
public class Servico extends ItemVendavel {
    @Column(name = "qtde_horas", nullable = false)
    private Double quantidadeHoras;

    public Servico() {
    }

    public Servico(String descricao, Double quantidadeHoras, Double valor) {
        super.setDescricao(descricao);
        this.quantidadeHoras = quantidadeHoras;
        super.setValorUnitario(valor);
    }

    public Double getQuantidadeHoras() { return quantidadeHoras; }
```



Façam o mapeamento de Cliente, Fornecedor

@MappedSuperclass

```
public abstract class Pessoa extends EntityId {
```

```
    @Column(name = "nome", nullable = false)
```

```
    private String nome;
```

```
    @Column(name = "telefone", nullable = false)
```

```
    private String telefone;
```

```
    @Column(name = "endereco", nullable = false)
```

```
    private String endereco;
```



```
    @Column(name = "email", nullable = false)
```

```
    private String email;
```

@Entity

public class Cliente extends Pessoa {

@Column(name = "cpf", nullable = false)

private String cpf;

@Column(name = "rg", nullable = false)

private String rg;

public String getCpf() { return cpf; }

public void setCpf(String cpf) { this.cpf = cpf; }

public String getRg() { return rg; }



```
@Entity
public class Fornecedor extends Pessoa {

    @Column(name = "cnpj", nullable = false)
    private String cnpj;
    @Column(name = "inscricao_estadual")
    private String inscricaoEstadual;

    public String getCnpj() { return cnpj; }

    public void setCnpj(String cnpj) { this.cnpj = cnpj; }

    public String getInscricaoEstadual() { return inscricaoEstadual; }
```

Ao rodar nossas estruturas de banco já são criadas



Vamos mapear nossas estruturas mais complexas

```
public class Compra extends EntityId implements OperacaoFinanceira {  
  
    private LocalDate dataCompra;  
    private Fornecedor fornecedor;  
    private String observacao;  
    private List<ItemCompra> itens = new ArrayList<>();  
  
    public LocalDate getDataCompra() { return dataCompra; }  
  
    public void setDataCompra(LocalDate dataCompra) { this.dataCompra = dataCompra; }
```


@Entity

```
public class Compra extends EntityId implements OperacaoFinanceira {
```

```
    @Column(name = "dt_compra")
```

```
    private LocalDate dataCompra;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "fornecedor_id")
```

```
    private Fornecedor fornecedor;
```

```
    @Column(name = "observacao")
```

```
    private String observacao;
```

```
    @OneToMany(mappedBy = "compra")
```

```
    private List<ItemCompra> itens = new ArrayList<>();
```

```
@Entity
public class ItemCompra extends EntityId {
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "produto_id")
```

```
    private Produto produto;
```

```
    @Column(name = "valor_unitario")
```

```
    private Double valorUnitario;
```

```
    @Column(name = "quantidade")
```

```
    private Double quantidade;
```

```
    @Column(name = "desconto")
```

```
    private Double desconto;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "compra_id")
```

```
    private Compra compra;
```

FK

```
    public ItemCompra() {
```

```
    }
```

Vamos mapear **Locação** agora, mesmo esquema

```
@Entity
public class Locacao extends EntityId implements OperacaoFinanceira {

    @Column(name = "dt_compra")
    private LocalDate dataLocacao;

    @Column(name = "dt_devolucao")
    private LocalDate dataDevolucao;

    @ManyToOne
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;

    @Column(name = "endereco")
    private String endereco;

    @Column(name = "observacao")
    private String observacao;

    @OneToMany(mappedBy = "locacao")
    private List<ItemLocacao> itens = new ArrayList<>();
}
```

```
@Entity
public class ItemLocacao extends EntityId {

    @ManyToOne
    @JoinColumn(name = "produto_id")
    private Produto produto;

    @Column(name = "valor_unitario")
    private Double valorUnitario;

    @Column(name = "quantidade")
    private Double quantidade;

    @Column(name = "desconto")
    private Double desconto;

    @ManyToOne
    @JoinColumn(name = "locacao_id")
    private Locacao locacao;

    public ItemLocacao() {
    }
}
```

Vamos mapear **Venda** agora, mesmo esquema

```
@Entity
public class Venda extends EntityId implements OperacaoFinanceira {

    @Column(name = "data_venda")
    private LocalDate dataVenda;

    @ManyToOne
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;

    @Enumerated(EnumType.STRING)
    @Column(name = "forma_pagamento")
    private FormaPagamento formaPagamento;

    @Column(name = "observacao")
    private String observacao;

    @OneToMany(mappedBy = "venda", cascade = CascadeType.ALL)
    private List<ItemVenda> itens = new ArrayList<>();
}
```

```
@Entity
public class ItemVenda extends EntityId{

    @ManyToOne
    @JoinColumn(name = "produto_servico_id")
    private ItemVendavel produtoServico;

    @Column(name = "valor_unitario", nullable = false)
    private Double valorUnitario;

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

    @Column(name = "desconto", nullable = false)
    private Double desconto;

    @ManyToOne
    @JoinColumn(name = "venda_id")
    private Venda venda;
```


Aqui faremos um novo **refactoring!**

@DiscriminatorValue

A anotação **@DiscriminatorValue** é usada em mapeamento de herança no JPA (Java Persistence API) para especificar o valor que será armazenado na coluna discriminadora (geralmente uma coluna em uma tabela) para identificar a subclasse a que uma entidade pertence em hierarquias de herança.

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "tipo_item")
public abstract class ItemVendavel extends EntityId {

    @Column(name = "descricao")
    private String descricao;

    @Column(name = "valor_unitario")
    private Double valorUnitario;

    @Column(name = "estocavel")
    private Boolean estocavel;

    public Boolean getEstocavel() { return estocavel; }
```



```
@Entity
@DiscriminatorValue("produto")
public class Produto extends ItemVendavel {

    @Column(name = "nome")
    private String nome;
    @Column(name = "preco_compra")
    private Double precoCompra;
    @Column(name = "dt_validade")
    private LocalDate dataValidade;
    @Column(name = "dt_prazo")
    private LocalDate dataPrazo;
    @Enumerated(EnumType.STRING)
    @Column(name = "status")
    private Status status;
```

```
@Entity
@DiscriminatorValue("servico")
public class Servico extends ItemVendavel {
    @Column(name = "qtde_horas")
    private Double quantidadeHoras;


    public Servico() {
    }

    public Servico(String descricao, Double quantidadeHoras, Double valor) {
        super.setDescricao(descricao);
        this.quantidadeHoras = quantidadeHoras;
        super.setValorUnitario(valor);
    }
}
```

Aqui faremos um novo **refactoring!**

Aqui faremos um novo **refactoring!**

```
public void addItemVenda(ItemVenda item) {  
    item.setVenda(this);  
    this.itens.add(item);  
}
```



<< Venda

```
public void addItemCompra(ItemCompra item) {  
    item.setCompra(this);  
    this.itens.add(item);  
}
```

<< Compra

```
public void addItemLocacao(ItemLocacao item) {  
    item.setLocacao(this);  
    this.itens.add(item);  
}
```

<< Locacao

No momento não faremos nada com a nossa classe **Balanço**, nessa classe utilizaremos uma abordagem diferente em um futuro próximo.

Beleza! Até aqui criamos toda a nossa estrutura de banco, mapeamos nossas entidades principais, agora deixaremos no SpringJPA criar nossa estrutura de tabelas rodando a aplicação novamente.

¹Em caso de **erro**, apague a estrutura de banco e rodem novamente

Agora vamos criar as estruturas que nos permitiram manipular os dados do nosso banco.

Criem o pacote *repository*

Criem a interface *ProdutoRepository*

```
package com.satc.satcloja.repository;

import com.satc.satcloja.model.Produto;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProdutoRepository extends JpaRepository<Produto, Long> {
}
```

A anotação **@Repository** faz parte do ecossistema Spring Framework e é usada para indicar que uma classe é um componente de repositório, ou seja, uma classe que lida com a persistência de dados, geralmente relacionada a acesso a banco de dados. Essa anotação é uma das três anotações de estereótipo no Spring, juntamente com @Service e @Controller, que são usadas para definir diferentes tipos de componentes em um aplicativo Spring.

A interface **JpaRepository** faz parte do Spring Data JPA, que é uma subprojeto da estrutura Spring que simplifica a implementação de repositórios JPA em seu aplicativo.

Operações Básicas de CRUD:



save: Salva ou atualiza uma entidade no banco de dados.

saveAll: Salva uma coleção de entidades.

findById: Busca uma entidade por ID.

existsById: Verifica se uma entidade com o ID especificado existe.

findAll(): Retorna todas as entidades no banco de dados.

findAllById: Retorna todas as entidades com IDs especificados.

count(): Retorna a contagem total de entidades.

Agora vamos testar essa BAGAÇA! =D

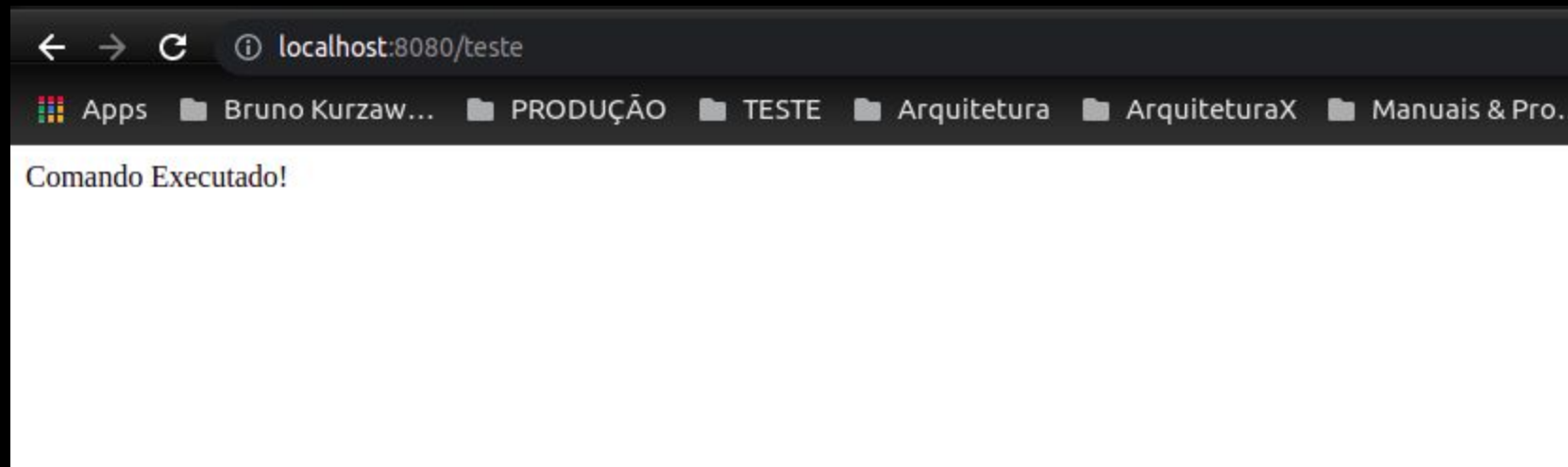
Para isso utilizaremos nossa classe de **HealthCheckController**

```
@GetMapping("/teste")  
public String healthCheck2() {  
  
    //Aqui vamos fazer nosso teste  
  
    return "Comando Executado!";  
}
```

```
@GetMapping("/teste")
public String healthCheck2() {

    Produto produto = new Produto();
    produto.setDescricao("Intel Pentium I5");
    produto.setNome("PC Intel");
    produto.setValorUnitario(1000.00);
    produto.setDataPrazo(LocalDate.now());
    produto.setDataValidade(LocalDate.now());
    produto.setPrecoCompra(850.00);
    produto.setStatus(Status.DISPONIVEL);
    produto.setEstocavel(Boolean.TRUE);

    return "Comando Executado!";
}
```

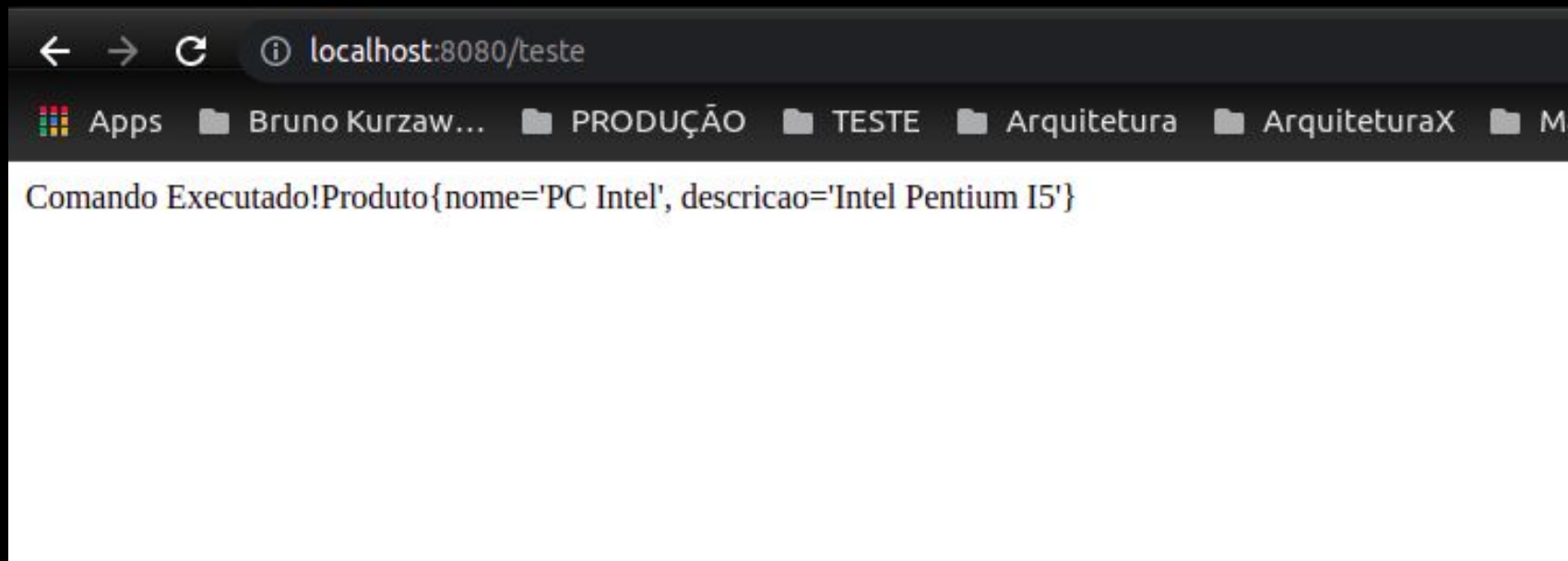


```
@GetMapping("/teste")
public String healthCheck2() {

    Produto produto = new Produto();
    produto.setDescricao("Intel Pentium I5");
    produto.setNome("PC Intel");
    produto.setValorUnitario(1000.00);
    produto.setDataPrazo(LocalDate.now());
    produto.setDataValidade(LocalDate.now());
    produto.setPrecoCompra(850.00);
    produto.setStatus(Status.DISPONIVEL);
    produto.setEstocavel(Boolean.TRUE);

    return "Comando Executado!" + produto;
}
```





Agora vamos fazer salvar em banco

```
@RestController
public class HealthCheckController {

    @Autowired
    public ProdutoRepository produtoRepository;
```



A anotação `@Autowired` faz parte do ecossistema Spring Framework e é usada para realizar injeção de dependência automática em componentes gerenciados pelo Spring. Essa anotação indica ao Spring para fornecer uma instância do bean (um objeto) correspondente a uma determinada classe ou interface e injetá-la automaticamente em um campo, método ou construtor de outra classe.

```
@GetMapping("/teste")
public String healthCheck2() {

    Produto produto = new Produto();
    produto.setDescricao("Intel Pentium I5");
    produto.setNome("PC Intel");
    produto.setValorUnitario(1000.00);
    produto.setDataPrazo(LocalDate.now());
    produto.setDataValidade(LocalDate.now());
    produto.setPrecoCompra(850.00);
    produto.setStatus(Status.DISPONIVEL);
    produto.setEstocavel(Boolean.TRUE);

    produtoRepository.save(produto);

    return "Comando Executado! " + produto.getId();
}
```





Criem os repository para **Cliente, Compra, Fornecedor, Locacao, Servico e Venda.**

```
Servico servico = new Servico();  
servico.setQuantidadeHoras(20.00);  
servico.setDescricao("Instalação Office");  
servico.setEstocavel(Boolean.TRUE);  
servico.setValorUnitario(150.00);  
  
servico = servicoRepository.save(servico);
```



```
Cliente cliente = new Cliente();  
cliente.setCpf("046969623");  
cliente.setRg("5224778");  
cliente.setNome("Bruno");  
cliente.setEmail("bruno.kurzawe@betha.com.br");  
cliente.setEndereco("Rua tal de tall");  
cliente.setTelefone("999089410");  
  
cliente = clienteRepository.save(cliente);
```

```
Venda venda = new Venda();
venda.setCliente(cliente);
venda.setDataVenda(LocalDate.now());
venda.setObservacao("Teste");
venda.setFormaPagamento(FormaPagamento.A_VISTA);

ItemVenda itemVenda = new ItemVenda(produto, 1000.00, 1.0, 10.00);
ItemVenda itemVenda2 = new ItemVenda(servico, 120.00, 1.0, 10.00);

venda.addItemVenda(itemVenda);
venda.addItemVenda(itemVenda2);

vendaRepository.save(venda);
```

Fim da aula 06...