Crie o banco de dados

Ex:

DROP DATABASE IF EXISTS cad\_vendas;

CREATE DATABASE IF NOT EXISTS cad\_vendas;

USE cad\_vendas;

CREATE TABLE funcionarios (

    id\_func BIGINT PRIMARY KEY AUTO\_INCREMENT,

    nome\_func VARCHAR(80) NOT NULL,

    matricula VARCHAR(80) NOT NULL UNIQUE,

    numero VARCHAR(20) NOT NULL,

    email VARCHAR(80) NOT NULL,

    desconto\_func INT NOT NULL

);

CREATE TABLE vendas (

    id\_venda BIGINT PRIMARY KEY AUTO\_INCREMENT,

    funcionario VARCHAR(80) NOT NULL,

    valor\_venda DOUBLE(10,2) NOT NULL,

    desconto\_venda DOUBLE(10,2)

);

-- Trigger para inserção

DELIMITER //

CREATE TRIGGER inserir\_desconto

BEFORE INSERT ON vendas

FOR EACH ROW

BEGIN

    -- Verifica se a variável @porcentagemDesconto está definida, caso contrário, define um valor padrão

    IF @porcentagemDesconto IS NULL THEN

        SET @porcentagemDesconto = 0; -- Define um valor padrão, se necessário

    END IF;

    SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01);

END //

DELIMITER ;

-- Trigger para atualização

DELIMITER //

CREATE TRIGGER atualizar\_desconto

BEFORE UPDATE ON vendas

FOR EACH ROW

BEGIN

    -- Verifica se a variável @porcentagemDesconto está definida, caso contrário, define um valor padrão

    IF @porcentagemDesconto IS NULL THEN

        SET @porcentagemDesconto = 0; -- Define um valor padrão, se necessário

    END IF;

    SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01);

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER atualizar\_funcionario\_vendas

BEFORE UPDATE ON funcionarios

FOR EACH ROW

BEGIN

    UPDATE vendas

    SET funcionario = NEW.nome\_func

    WHERE funcionario = OLD.nome\_func;

END //

DELIMITER ;

Execute no mysql

Ao começar um projeto, você precisa ir no site e fazer algumas configurações

<https://start.spring.io/>

ao iniciar o projeto vá nos arquivos pom.xml e coloque essas dependências:

<dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-thymeleaf</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-devtools</artifactId>

            <scope>runtime</scope>

            <optional>true</optional>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-test</artifactId>

            <scope>test</scope>

        </dependency>

        <dependency>

            <groupId>nz.net.ultraq.thymeleaf</groupId>

            <artifactId>thymeleaf-layout-dialect</artifactId>

            <version>3.2.0</version>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-data-jpa</artifactId>

        </dependency>

        <dependency>

            <groupId>com.mysql</groupId>

            <artifactId>mysql-connector-j</artifactId>

            <scope>runtime</scope>

        </dependency>

    </dependencies>

Atualize o maven

**Configure a conexão mysql – application.properties**

spring.datasource.url=jdbc:mysql://host\_ip/nome\_banco?createDatabaseIfNotExist=true  
spring.datasource.username=usuario\_banco  
spring.datasource.password=senha  
  
  
spring.mvc.hiddenmethod.filter.enabled=true

createDatabaseIfNotExist=true

se o banco não existir, ele cria

spring.mvc.hiddenmethod.filter.enabled=true

serve para ativar rotas ocultas, put ou até mesmo formulários dentro de formulários

inicie a aplicação

entre em localhost:8080 ou outra porta configurada

ex:  
spring.datasource.url=jdbc:mysql://localhost:8081/cad\_vendas?createDatabaseIfNotExist=true

url : localhost:8080

**Vamos trabalhar com o modelo MVC (Model Vision Controller)**

**Controller:**

Crie um package controller no package principal;

Crie uma classe;

package com.crud.cad\_usuario.controller;  
  
public class CrudController {  
}

vamos adicionar as anotações para que o Spring saiba que é um controller e ajude as definições de http

@Controller // é uma anotação usada para marcar uma classe como um controlador no contexto do Spring MVC. Essa anotação indica que a classe é responsável por lidar com as requisições HTTP e retornar as respostas correspondentes.  
@RequestMapping // usada para mapear uma requisição HTTP para um método específico dentro de um controlador.   
public class CrudController {  
}

@GetMapping("/")

**Usado para criar uma rota, no caso será a pagina principal**

@GetMapping("/Exemplo/pagina")

**Ex:** localhost:8080/Exemplo/pagina

Em baixo vai o método que vai rodar a página:

Esse método vai retornar uma String que será onde o arquivo HTML está

@GetMapping("/")  
public String paginaPricipal(){  
 return "";  
}

**Vision:**

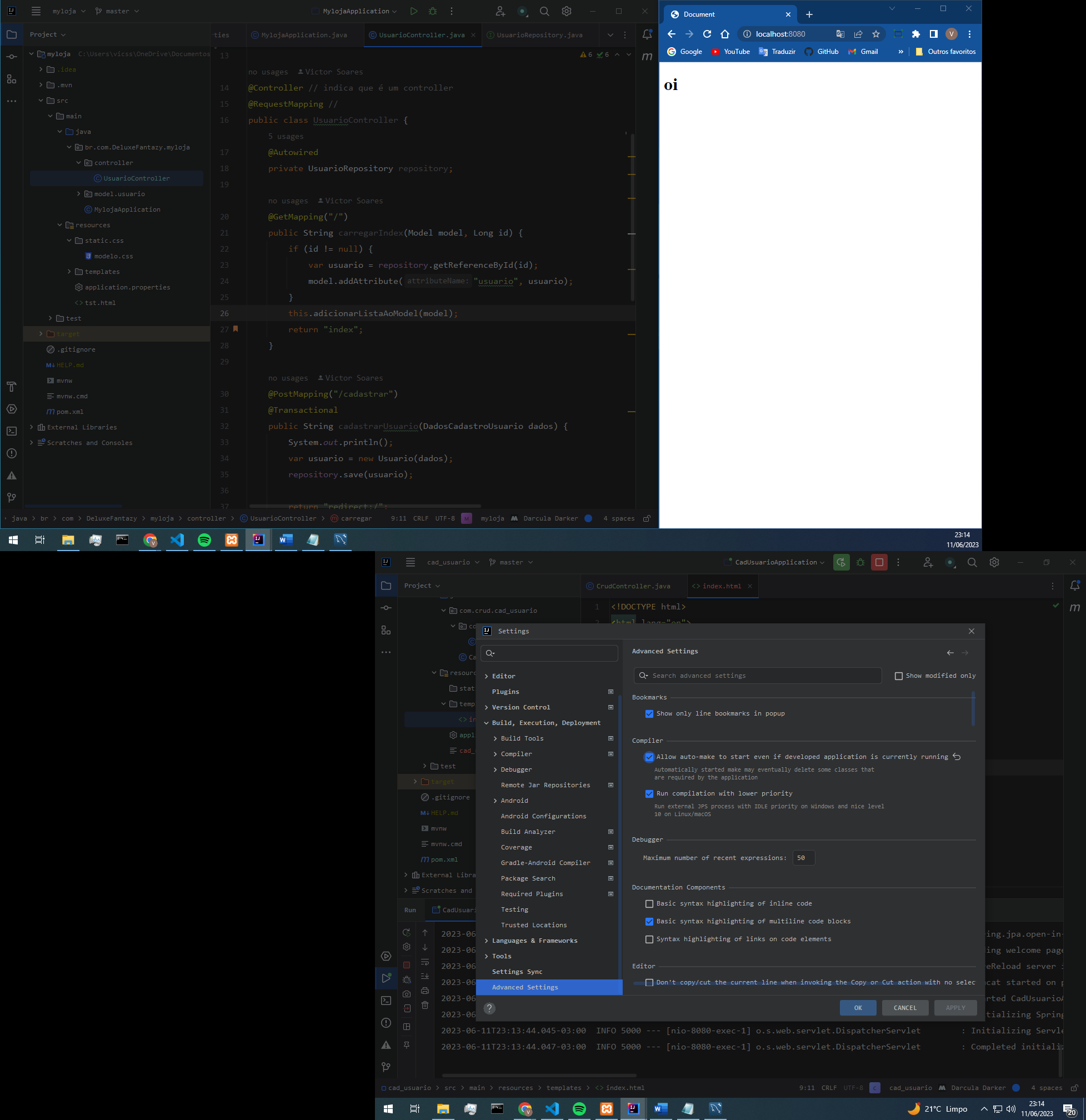
No maven, a visão ficará na pasta resources qaue tem duas pastas:

Static: ficará os arquivos estáticos como CSS e JS

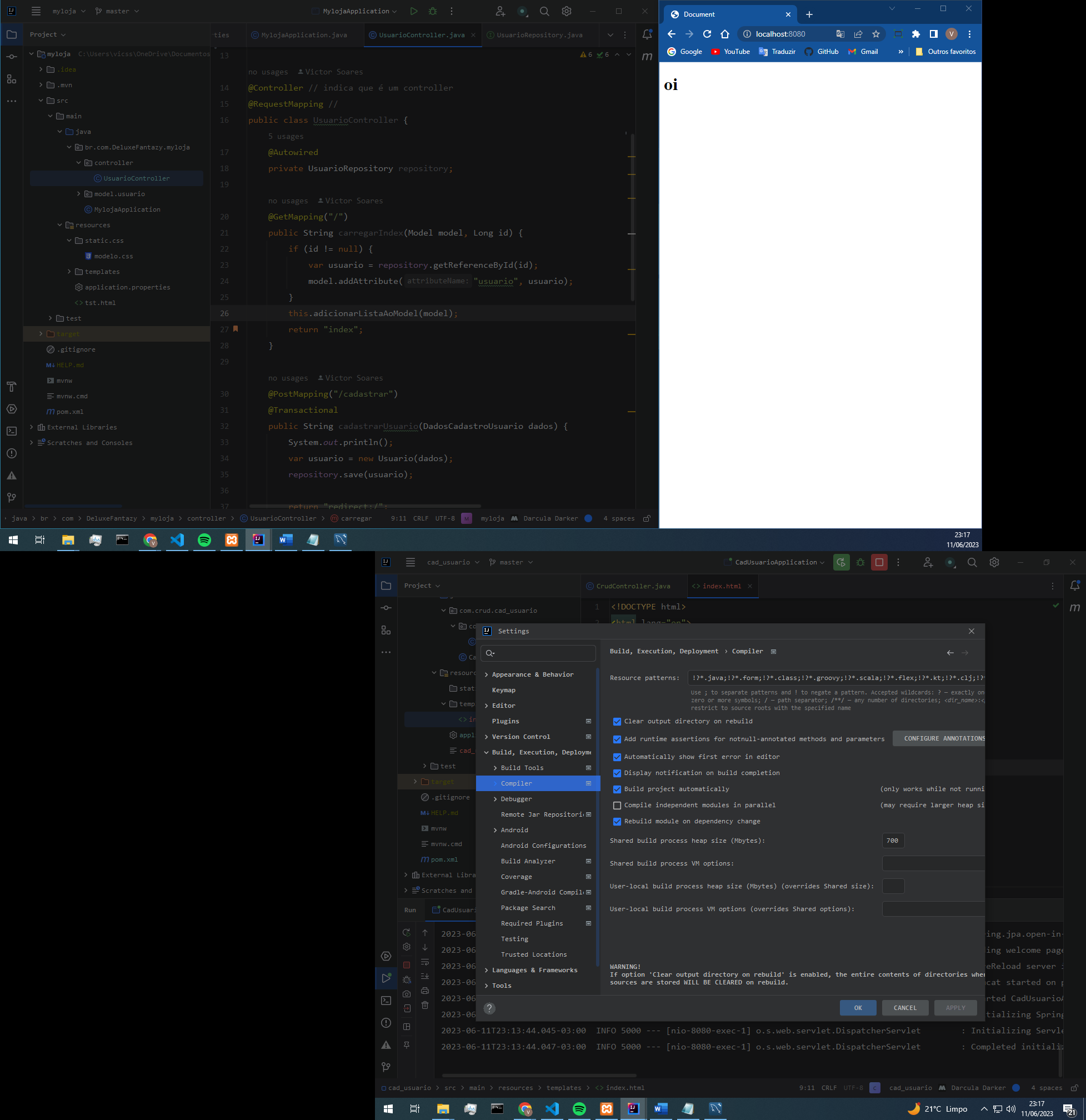
Templates: os arquivos HTML

**EXTRA (Vamos configurar o devTolls do IntelliJ para não precisar ficar parando e iniciando a aplicação)**

1. vá em configurações
2. Advanced Settings



1. Volte para configurações
2. Build, Execution, Deployment
3. Compiler



**// ---------------------------------------------------------------------------------**

**Criando um arquivo na pasta templetes**

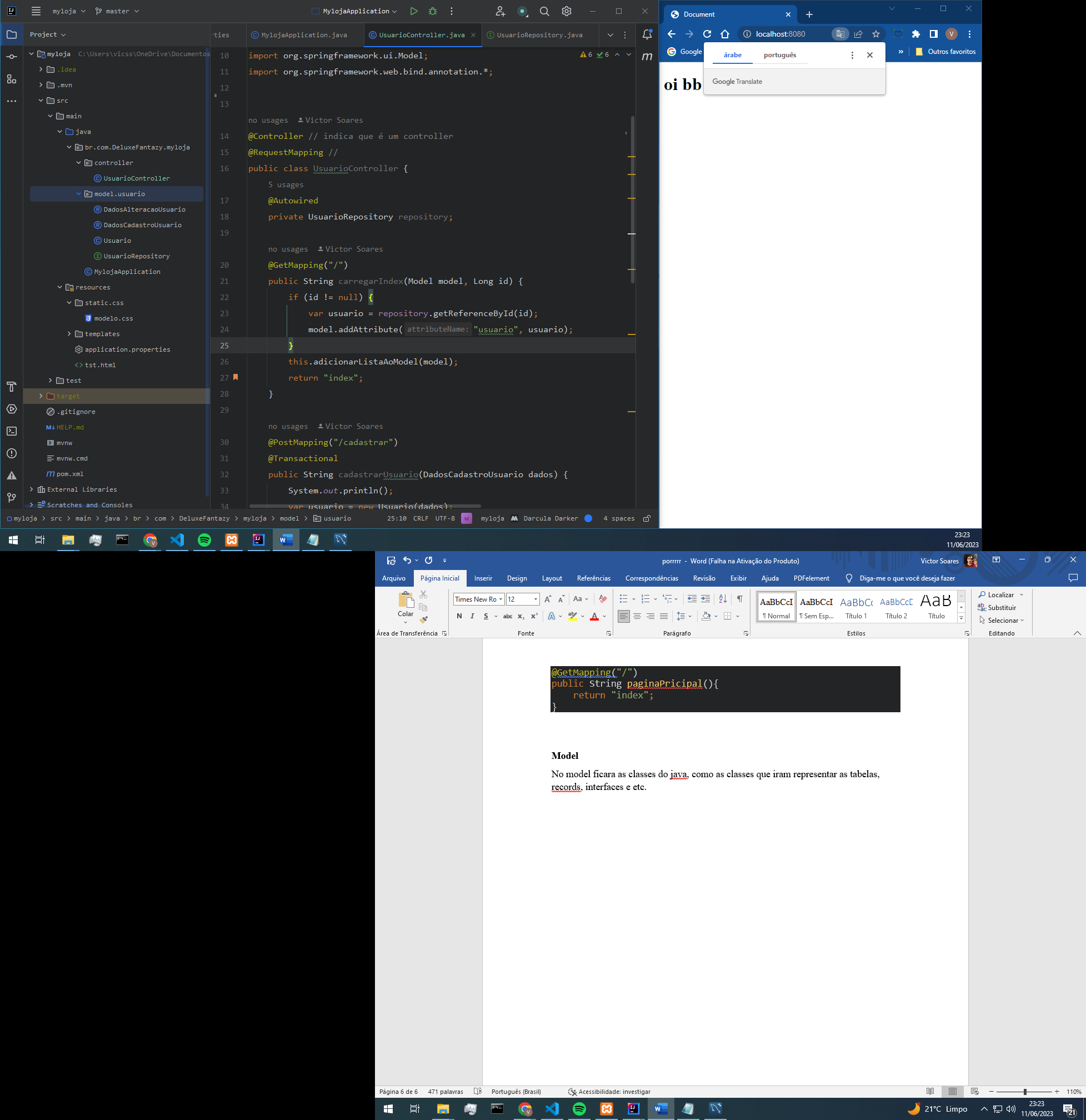
**Ex:** index.html

@GetMapping("/")  
public String paginaPricipal(){  
 return "index";  
}

**Model**

No model ficara as classes do java, como as classes que iram representar as tabelas, records, interfaces e etc.

Ex:



**Usando as dependências thymeleaf e ultraThymeleaf**

para usa o thymeleaf é preciso chamar ele no html

<html lang="pt-br"  
 xmlns:th="http://thymeleaf.org"  
 xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"  
>

xmlns:nome\_variavel=”link\_da\_dependencia”

**Usando a função Template do ultraThymeleaf**

Essa dependência permite que o projeto use uma página principal que vai em todas as páginas, como uma informação ou css padrão

Vamos criar uma página na pasta templates/layouts

Ex: layout.html:

<!DOCTYPE *html*>

<html *lang*="pt-br"

*xmlns:th*="http://thymeleaf.org"

*xmlns:layout*="http://www.ultraq.net.nz/thymeleaf/layout"

>

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Vendas</title>

</head>

<body>

    <div *layout:fragment*="conteudo"></div>

</body>

</html>

* Adicionamos os links das dependências
* Vamos usar a variável layout
* Usamos a tag :fragment”nomeDaSuaPreferencia”
* Dentro dessadiv vai as outras páginas

Ex:

<body>

    <h1>oi</h1>

    <div *layout:fragment*="conteudo"></div>

</body>

* Em todas as páginas vai está escrito oi

Como chamar uma página para esse layout

Ex: index.html

<div  
 layout:decorate="~{layouts/layout.html}"  
 layout:fragment="conteudo"  
>  
 <h1>Conteudo do index</h1>  
</div>

**Obs;** layout(o mesmo nome que você colocou no tempalte)

Usamos as tags:

layout:decorate=”diretório\_do\_arquivo”

layout:fragmente=”mesmo\_nome\_do\_fragment\_do\_template”

**thymeleaf:**

como incluir páginas usando thymeleaf

crie uma pasta fragments

la você vai crias todas as paginas, como formulários, listas e etc.

ex: fragments/FormFunc.html

<div *th:fragment*="FormFunc">

    <form *action*="">

        <table>

            <thead>

                <tr>

                    <th>Nome</th>

                    <th>Email</th>

                </tr>

            </thead>

            <tbody>

                <tr>

                    <td></td>

                    <td></td>

                </tr>

            </tbody>

        </table>

    </form>

</div>

th:fragment=”nome”

onde vai ser incluido :

<div

*layout:decorate*="~{layouts/layout.html}"

*layout:fragment*="conteudo"

>

    <h1>Conteudo do index</h1>

    <div *th:replace*="fragments/FormFunc :: FormFunc"></div>

</div>

th:replace=”diretorio\_do\_arquivo :: nome\_fragment”

**Vamos ao CRUD**

**Cadastro:**

**FormFunc.html:**

<div *th:fragment*="FormFunc">

    <form *action*="">

        <table>

            <thead>

                <tr>

                    <th>Funcionario</th>

                </tr>

            </thead>

            <tbody>

                <tr>

                    <td><label *for*="nomeFunc">Nome:</label></td>

                    <td><input *type*="text" *name*="nomeFunc" *id*="nomeFunc"></td>

                </tr>

                <tr>

                    <td><label *for*="matricula">Matrícula:</label></td>

                    <td><input *type*="text" *name*="matricula" *id*="matricula"></td>

                </tr>

                <tr>

                    <td><label *for*="numero">Número:</label></td>

                    <td><input *type*="text" *name*="numero" *id*="numero"></td>

                </tr>

                <tr>

                    <td><label *for*="email">Email:</label></td>

                    <td><input *type*="text" *name*="email" *id*="email"></td>

                </tr>

                <tr>

                    <td><label *for*="descontoFunc">Desconto:</label></td>

                    <td>

                        <select *name*="descontoFunc" *id*="descontoFunc">

                            <option *th:each*="i : ${#numbers.sequence(5, 100, 5)}" *th:value*="${i}" *th:text*="${i + '%'}"></option>

                        </select>

                    </td>

                </tr>

                <tr><td *colspan*="2"><button *type*="submit">Enviar</button></td></tr>

            </tbody>

        </table>

    </form>

</div>

<option *th:each*="i : ${#numbers.sequence(5, 100, 5)}" *th:value*="${i}" *th:text*="${i + '%'}"></option>

**Laços usando thymeleaf:**

Th:each=” i : ${#numbers.sequence(inicio, fim, adição)}”

Th:text”imprme no option”

**Crie um package com o nome da tabela em model:**

**Funcionario.class:**

package com.crud.cad\_usuario.model.funcionario;  
  
  
import jakarta.persistence.\*;  
  
@Entity // usado para o JPA reconhecer essa classe e mapela para o baanco de dados  
@Table(name="funcionarios")// indica qual é a tabela  
public class Funcionario {  
  
 @Id // indica que esse atributo é relacionado a chave Primária  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*) // indica que a chave é auto\_increment  
 private long idFunc;  
 private String nomeFunc;  
 private String matricula;  
 private String numero;  
 private String email;  
 private int descontoFunc;  
  
 public Funcionario (DadosCadastroFuncionario dados) { // vai usar um record, que é feito especiamente para esses cados de pegar as infomações dos formularios e fazerem toda a conversão para o objeto  
 this.nomeFunc = dados.nomeFunc();  
 this.matricula = dados.matricula();  
 this.numero = dados.numero();  
 this.email = dados.email();  
 this.descontoFunc = dados.descontoFunc();  
 }  
  
 public Funcionario () {} // é importante ter esse contrutor padrão, se não vai dar erro mais na frente  
  
 public long getIdFunc() {  
 return idFunc;  
 }  
  
 public String getNomeFunc() {  
 return nomeFunc;  
 }  
  
 public String getMatricula() {  
 return matricula;  
 }  
  
 public String getNumero() {  
 return numero;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public int getDescontoFunc() {  
 return descontoFunc;  
 }  
  
 @Override  
 public String toString() { // retorna uma string com as informções de cada atributo  
 return "Funcionario{" +  
 "idFunc=" + idFunc +  
 ", nomeFunc='" + nomeFunc + '\'' +  
 ", matricula='" + matricula + '\'' +  
 ", numero='" + numero + '\'' +  
 ", email='" + email + '\'' +  
 ", descontoFunc=" + descontoFunc +  
 '}';  
 }  
  
 public void atualizaDados(DadosAlteracaoFuncionario dados) {  
 this.nomeFunc = dados.nomeFunc();  
 this.matricula = dados.matricula();  
 this.numero = dados.numero();  
 this.email = dados.email();  
 this.descontoFunc = dados.descontoFunc();  
 }  
}

package com.crud.cad\_usuario.model.funcionario;  
  
public record DadosCadastroFuncionario(String nomeFunc, String matricula, String numero, String email, int descontoFunc) {  
} // vai pegar as informações dos inputs e pode ser usado quando quiser, emqualquer classe. obs: o nome das variáveis tem que ser os mesmos do input

package com.crud.cad\_usuario.model.funcionario;  
  
public record DadosAlteracaoFuncionario(Long idFunc, String nomeFunc, String matricula, String numero, String email, int descontoFunc) {  
} // a mesama coisa que cadastro, mas também vai pegar as informações do id

**Obs:** O Java e sensity Key, ou seja ele diferencia caracteres maiúsculos de minúsculos, já o mysql não, então quando o jpa for fazer a conversão, no java vai estar nomeFunc, mas a stantment que a jpa vai usar é nome\_func, então o nome no sql deve estar igual no stantment (nome\_func)

No controller

@Autowired // Ele permite que você injete instâncias de classes em outras classes, evitando a necessidade de criar manualmente as instâncias.

private FuncionarioRepository repository; // vai buscar a interface e usala como atributo

Interface

package com.crud.cad\_usuario.controller;  
  
import com.crud.cad\_usuario.model.funcionario.Funcionario;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface FuncionarioRepository extends JpaRepository<Funcionario, Long> {} // vai estender a uma classe do jpa, Funcionario- é o objeto representado pela tabela, e o Long é o tipo da chave primária

Controller

@GetMapping("/funcionarios") // criando um link --> localhost:8080/funcionarios  
public String paginaFunconarios(){  
 return "subPage/funcionarios";  
}  
  
@PostMapping("/cadastrar") // mapeia uma requisição http do tipo Post  
@Transactional // garante que a operação de salvar o funcionário no banco de dados seja realizada dentro de uma transação.  
public String cadFuncionario(DadosCadastroFuncionario dados) { // vai receber as infromações via post e mandalas para o record  
 var funcionarios = new Funcionario(dados); // vai usar o construtor que muda os dados, usando os dados do record  
 repository.save(funcionarios); // vai salvar as informações do objeto em um método do jpa que deve ser um insert into e executalo no banco  
 return "redirect:/funcionarios"; // redireciona a página  
}

**Read:**

<div *th:fragment*="ListFunc">

    <table>

        <thead>

            <tr>

                <th>Nome</th>

                <th>Matrícula</th>

                <th>Número</th>

                <th>Email</th>

                <th>Desconto de Funcionário</th>

                <th *colspan*="2">Ações</th>

            </tr>

        </thead>

        <tbody>

            <tr *th:each*="lista : ${funcionarios}">

                <td *th:text*="${lista.nomeFunc}"></td>

                <td *th:text*="${lista.matricula}"></td>

                <td *th:text*="${lista.numero}"></td>

                <td *th:text*="${lista.email}"></td>

                <td *th:text*="${lista.descontoFunc}"></td>

            </tr>

        </tbody>

    </table>

</div>

th:each=”nomeVariavelLoop : ${variavelAtributo}”

th:text=”${variavelLoop.atributoObejeto}”

controller:

private void listaModel(Model model, String atributoWeb) { // a classe Model é usada para transportar as informações do banco para um array que poderá ser mostrado na web  
 model.addAttribute(atributoWeb, repository.findAll()); // vai levar para a variável atributoWeb no html as informações do objeto que no caso é um método da jpa, que deve usar um SELECT \*  
 }  
}

@GetMapping("/funcionarios") // criando um link --> localhost:8080/funcionarios  
public String paginaFunconarios(Model model){  
 this.listaModel(model, "funcionarios"); // chama o método para enviar as informações para variavel web  
 return "subPage/funcionarios";  
}

**Delete**

**Lista.html**

<td><a *th:href*="@{/delFunc?idFunc={id}(id=${lista.idFunc})}"><button>Deletar</buteton></a></td>

@{} – é preciso para que possam ser adicionadas variáveis do thymeleaf ao link

th:href=”@{/pagina?nomeParametro={variavel}(variável=${varLoop.colunatabela})}”

@GetMapping("/delFunc")  
public String delFuncionarios (Long idFunc) { // vai pegar a informação que estiver depois do parâmetro idFunc  
 repository.deleteById(idFunc); // deletar por id  
 return "redirect:/funcionarios";  
}

**Update**

<td><a *th:href*="@{/funcionarios?idFunc={id}(id=${lista.idFunc})}"><button>Alterar</button></a></td>

@GetMapping("/funcionarios") // criando um link --> localhost:8080/funcionarios  
public String paginaFuncionarios(Model model, Long idFunc){ // pega o id pelo link  
 if(idFunc != null) { // se o id for diferente de null, no cado da alteração  
 var funcionario = repository.getReferenceById(idFunc); // ele busca as informações por id  
 model.addAttribute("funcionario", funcionario); // joga essas inormações numa variavel web  
 }  
 this.listaModel(model, "funcionarios"); // chama o método para enviar as informações para variavel web  
 return "subPage/funcionarios";  
}

Form.html (o mesmo do cadastro)

<div *th:fragment*="FormFunc">

    <form *action*="/cadastrar" *method*="post">

        <table>

            <thead>

                <tr>

                    <th>Funcionario</th>

                </tr>

            </thead>

            <tbody>

                <input *type*="hidden" *name*="\_method" *value*="PUT" *th:if*="${funcionario != null}">

                <input *type*="hidden" *name*="idFunc" *th:value*="${funcionario?.idFunc}">

                <tr>

                    <td><label *for*="nomeFunc">Nome:</label></td>

                    <td><input *type*="text" *name*="nomeFunc" *id*="nomeFunc" *th:value*="${funcionario?.nomeFunc}"></td>

                </tr>

                <tr>

                    <td><label *for*="matricula">Matrícula:</label></td>

                    <td><input *type*="text" *name*="matricula" *id*="matricula" *th:value*="${funcionario?.matricula}"></td>

                </tr>

                <tr>

                    <td><label *for*="numero">Número:</label></td>

                    <td><input *type*="text" *name*="numero" *id*="numero" *th:value*="${funcionario?.numero}"></td>

                </tr>

                <tr>

                    <td><label *for*="email">Email:</label></td>

                    <td><input *type*="text" *name*="email" *id*="email" *th:value*="${funcionario?.email}"></td>

                </tr>

                <tr>

                    <td><label *for*="descontoFunc">Desconto:</label></td>

                    <td>

                        <select *name*="descontoFunc" *id*="descontoFunc" *th:value*="${funcionario?.descontoFunc}">

                            <option *th:each*="i : ${#numbers.sequence(5, 100, 5)}" *th:value*="${i}" *th:text*="${i + '%'}"></option>

                        </select>

                    </td>

                </tr>

                <tr><td *colspan*="2"><button *type*="submit">Enviar</button></td></tr>

            </tbody>

        </table>

    </form>

</div>

1-

<input *type*="hidden" *name*="\_method" *value*="PUT" *th:if*="${funcionario != null}">

Vai enviar uma requisição PUT para o controller. **Obs:** essa requisição tem que ter o mesmo direcionamento que o cadastrar

*th:if*="${funcionario != null}">

cria um if que vai checar se a variável funcionario é nula

se for, basta colocar um ? depois da variável que ela vai ignorar o código

se não colocar o ?, vai dar erro na página

*th:value*="${variavelModel?.colunaTabela}"

ex:

*th:value*="${funcionario?.nomeFunc}"

<select *name*="descontoFunc" *id*="descontoFunc" *th:value*="${funcionario?.descontoFunc}">

                            <option *th:value*="${funcionario?.descontoFunc}" *th:text*="${funcionario?.descontoFunc}"*selected*></option>

                            <option *th:each*="i : ${#numbers.sequence(5, 100, 5)}" *th:value*="${i}" *th:text*="${i + '%'}"></option>

                        </select>

Th:each=”variável : ${#numbres.sequence(numInicio, numFim, incrementoNum) 🡪 comando do thymeleaf para fazer uma contagem }”

@PutMapping("/cadastrar") // tem que ter o mesmo destino que o de cadastrar  
@Transactional  
public String altFuncionario (DadosAlteracaoFuncionario dados) {// vai pegar as informações do record alterarDados  
 var funcionario = repository.getReferenceById(dados.idFunc()); // vai pegar as informações da tabela baseado no id  
 funcionario.atualizaDados(dados); // usa um método jpa para atualizar os dados  
 return "redirect:/funcionarios";  
}

Pronto. Todos os métodos do crud foram aplicados!!!

**Tabela Vendas**

Na tabela vendas adicionamos algumas funcionalidades interessantes

<input *type*="hidden" *name*="\_method" *value*="PUT" *th:if*="${venda != null}">

                <input *type*="hidden" *name*="idVenda" *th:value*="${venda?.idVenda}">

                <tr>

                    <td><label *for*="funcionario">Funcionario:</label></td>

                    <td>

                        <select *name*="funcionario" *id*="selectFunc">

                            <option *th:value*="${venda?.funcionario}" *th:text*="${venda?.funcionario}"*selected*></option>

                            <option *th:each*="lista : ${funcionarios}" *th:value*="${lista.nomeFunc}" *th:text*="${lista.nomeFunc}"></option>

                        </select>

                    </td>

                </tr>

                <tr>

                    <td><label *for*="valorVenda">Valor Venda:</label></td>

                    <td><input *type*="text" *name*="valorVenda" *id*="valorVenda" *th:value*="${venda?.valorVenda}"></td>

                </tr>

                <tr><td *colspan*="2"><button *type*="submit" *onclick*="teste()">Enviar</button></td></tr>

A começar pelo formulario, adicionamos um for each no select que busca o nome dos funcionarios

@GetMapping("/vendas") // criando um link --> localhost:8080/vendas  
public String paginaVendas(Model model, Long idVenda){ // pega o id pelo link  
  
 if(idVenda != null) { // se o id for diferente de null, no cado da alteração  
 var venda = repositoryVenda.getReferenceById(idVenda); // ele busca as informações por id  
 model.addAttribute("venda", venda); // joga essas inormações numa variavel web  
 }  
 this.listaModelFunc(model,"funcionarios"); // para o select dos nomes dos funcionarios  
 this.listaModelVenda(model, "vendas"); // chama o método para enviar as informações para variavel web  
 return "subPage/vendas";  
}

Vamos a coluna Desconto Vendas

Ela não e adicionada nos formulários

Ela vai pegar a porcentagem de desconto da tabela funcionário, fazer um calculo e adicionar na tabela vendas

Para isso vamos enviar uma variável para o mysql pelo comando

SET @porcentagemDesconto = ?

Para isso precisaremos criar uma nova classe em outro package (DAO)

public abstract class CrudDAO

como ela não é uma entidade, e não terá atributos, elá e abstrata com métodos estáticos

public static void mudarPorcentagem(EntityManager entityManager, int porcentagemDesconto) {  
 String sql = "SET @porcentagemDesconto = ?"; // a query que vai se enviada para o banco  
 Query nativeQuery = entityManager.createNativeQuery(sql); // Query --> classe do jpa que permite execuções de querys no db, esse comando vai preparar a string que contém a query  
 nativeQuery.setParameter(1, porcentagemDesconto); // vai mudar o ? pelo parametro int  
 nativeQuery.executeUpdate(); // executa a query no banco de dados   
}

no controller vamos criar um novo atributo

@Autowired  
private EntityManager entityManager; // classe do Jpa usada para mapear objetos para tabelas do banco de dados

Pra que serve esse método? Vai mudar a variável @porvcentagemDesconto no bd que vai ser usado como infomação numa trigger

-- Trigger para inserção

DELIMITER //

CREATE TRIGGER inserir\_desconto

BEFORE INSERT ON vendas

FOR EACH ROW

BEGIN

    -- Verifica se a variável @porcentagemDesconto está definida, caso contrário, define um valor padrão

    IF @porcentagemDesconto IS NULL THEN

        SET @porcentagemDesconto = 0; -- Define um valor padrão, se necessário

    END IF;

    SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01);

END //

DELIMITER ;

Ao inserir um dado na tabela vendas, depois ele vai pegar esse valor da venda e vai multiplicar pela porcentagem calculada e colocar o dado na coluna desconto\_venda

-- Trigger para atualização

DELIMITER //

CREATE TRIGGER atualizar\_desconto

BEFORE UPDATE ON vendas

FOR EACH ROW

BEGIN

    -- Verifica se a variável @porcentagemDesconto está definida, caso contrário, define um valor padrão

    IF @porcentagemDesconto IS NULL THEN

        SET @porcentagemDesconto = 0; -- Define um valor padrão, se necessário

    END IF;

    SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01);

END //

DELIMITER ;

A mesma coisa, só que para alteração

Mas qual desconto aplicar se na tabela tem vários funcionários, o nome enviado pelo formulário será salvo e enviado para outro método do DAO

public static int consultarDesconto(EntityManager entityManager, String nomeFuncionario) {  
 String sql = "SELECT desconto\_func FROM funcionarios WHERE nome\_func = ?"; // selecionando o desconto\_func com base no nome  
 Query query = entityManager.createNativeQuery(sql);  
 query.setParameter(1, nomeFuncionario);  
 Object result = query.getSingleResult(); // retorna o dado em formato de Objeto  
 return result != null ? (int) result : 0; // aqui fazemos um casting explicito transfomando ele em int  
 // é um operador ternario que retorna informações a base de condições  
 // condição ? valor\_se\_verdadeiro : valor\_se\_falso;  
}

no controller

@PostMapping("/cadVenda") // mapeia uma requisição http do tipo Post  
@Transactional // garante que a operação de salvar o funcionário no banco de dados seja realizada dentro de uma transação.  
public String cadVenda(DadosCadastroVenda dados) { // vai receber as infromações via post e mandalas para o record  
 int desconto = CrudDAO.*consultarDesconto*(entityManager, dados.funcionario()); // aqui buscamos o desconto\_func  
 CrudDAO.*mudarPorcentagem*(entityManager, desconto); // aqui mudamos a porcentagem

A trigger vai fazer o resto

Faça a mesma coisa no alterarVenda

**Página Soma de Valores**

Nessa página tem um formulário e uma lista

O formulário escolhe o nome do funcionário

E a lista retorna quais vendas ele fez e o total de tudo

<form *action*="/buscaFunc" *method*="post" *id*="test">

            <table>

                <thead><tr><th *colspan*="2">Soma de Vendas Por Funcionario</th></tr></thead>

                <tbody>

                    <tr>

                        <td><label *for*="funcionario">Funcionario:</label></td>

                        <td>

                            <select *name*="funcionario" *id*="selectFunc">

                                <option *value*="Selecione" *selected* *disabled*>Selecione</option>

                                <option *th:each*="lista : ${funcionarios}" *th:value*="${lista.nomeFunc}" *th:text*="${lista.nomeFunc}"></option>

                            </select>

                        </td>

                    </tr>

                </tbody>

            </table>

            <div *id*="divBtn">

                <button *type*="submit">Enviar</button>

            </div>

        </form>

<table>

        <thead>

            <tr>

                <th>Vendas</th>

                <th>Descontos</th>

            </tr>

        </thead>

        <tbody>

            <tr *th:each*="lista : ${vendas}">

                <td *th:text*="${lista.valorVenda}"></td>

                <td *th:text*="${lista.descontoVenda}"></td>

            </tr>

        </tbody>

        <tfoot>

            <tr>

                <th *colspan*="2">

                    Total

                </th>

            </tr>

            <tr *th:each*="lista : ${somaValores}">

                <td *th:text*="${lista.valorVenda}"></td>

                <td *th:text*="${lista.descontoVenda}"></td>

            </tr>

        </tfoot>

    </table>

Em DAO

public static Set<SomaValores> obterValoresFunc(EntityManager entityManager, String nomeFuncionario) {  
 String sql = "SELECT funcionario, valor\_venda, desconto\_venda FROM vendas WHERE funcionario = ?"; // seleciona todos os valores e descontos pelo nome\_func  
 Query query = entityManager.createNativeQuery(sql);  
 query.setParameter(1, nomeFuncionario);  
 List<Object[]> resultList = query.getResultList(); // retorna o resultado numa classe de array (List)  
  
 Set<SomaValores> resultSet = new HashSet<>(); // cria um array Set com base nas informações do record  
 for (Object[] result : resultList) { // cria um Obejct para representar o set no foreach  
 String funcionario = (String) result[0]; // cria variveis e guarda as informações nelas  
 BigDecimal valorVenda = BigDecimal.*valueOf*((Double) result[1]); // pegar um valor decimal e guarda num Bigdecimal atravé de um casting  
 BigDecimal descontoVenda = BigDecimal.*valueOf*((Double) result[2]);  
  
 SomaValores somaValores = new SomaValores(funcionario, valorVenda, descontoVenda); // salva as informações das variveis no record  
 resultSet.add(somaValores); // adiciona na lista Set  
 }  
  
 return resultSet; // retorna alista do tipo Set

public static Set<SomaValores> obterSomaVendasDescontos(EntityManager entityManager, String nomeFuncionario) {  
 String sql = "SELECT funcionario, SUM(valor\_venda), SUM(desconto\_venda) FROM vendas WHERE funcionario = ? GROUP BY funcionario"; // seleciona a soma dos valores com base no nome\_func  
 Query query = entityManager.createNativeQuery(sql);  
 query.setParameter(1, nomeFuncionario);  
 List<Object[]> resultList = query.getResultList();  
  
 Set<SomaValores> resultSet = new HashSet<>();  
 for (Object[] result : resultList) {  
 String funcionario = (String) result[0];  
 BigDecimal valorVenda = BigDecimal.*valueOf*((Double) result[1]);  
 BigDecimal descontoVenda = BigDecimal.*valueOf*((Double) result[2]);  
  
 SomaValores somaValores = new SomaValores(funcionario, valorVenda, descontoVenda);  
 resultSet.add(somaValores);  
 }  
  
 System.*out*.println(resultSet);  
  
 return resultSet;  
}

em Controller

@GetMapping("/somaValores")  
@Transactional  
public String paginaValores(Model model, String func) { // recebe a variavel do post pelo link --> obs: tem que se o mesmo nome que está no link  
 this.listaModelFunc(model, "funcionarios"); // retorna os nome dos funcionarios no formulario  
 Set<SomaValores> listaValores = CrudDAO.*obterValoresFunc*(entityManager, func); // obtem valores do funcionario  
 Set<SomaValores> soma = CrudDAO.*obterSomaVendasDescontos*(entityManager, func);  
 model.addAttribute("vendas", listaValores); // mostra os valores na lista, com a varivel vendas para web  
 model.addAttribute("somaValores", soma); // mostra a uma atravez da variavel somaValores na web  
 return "subPage/somaValores";  
}  
  
@PostMapping("/buscaFunc")  
@Transactional  
public String buscaFuncionario(String funcionario) { // ao clicar em enviar ele recebe a fincomação do input e guarda na string funcionario  
 return "redirect:/somaValores?func=" + funcionario; // envia a variavel funcionario pelo get  
}  
// ao clicar em enviar ele recebe a fincomação do input e guarda na string funcionario

**EXTRA**

Como adicionar aquivos estáticos no projeto?  
aqui está

Js:

<script *th:src*="@{/script/modelo.js}"></script>

css

<link *rel*="stylesheet" *href*="../../static/css/FormSomaValores.css" *th:href*="@{/css/FormSomaValores.css}">

DELIMITER //

CREATE TRIGGER atualizar\_funcionario\_vendas

BEFORE UPDATE ON funcionarios

FOR EACH ROW

BEGIN

    UPDATE vendas

    SET funcionario = NEW.nome\_func

    WHERE funcionario = OLD.nome\_func;

END //

DELIMITER ;

Essa trigger permite que quando você altere o nome do funcionário na tabela funcionário, todos os registros que tiverem esse nome na coluna funcionário da tabela vendas também será mudado, dando a semântica de que os dois estão conectados e são a mesma pessoa

**FIM**

**Algumas alterações no banco de dados**DROP DATABASE IF EXISTS cad\_vendas; -- se o database existir, exclua

CREATE DATABASE IF NOT EXISTS cad\_vendas; -- se o database não existir, crie

USE cad\_vendas; -- usar database

CREATE TABLE funcionarios (

    id\_func BIGINT PRIMARY KEY AUTO\_INCREMENT,

    nome\_func VARCHAR(80) NOT NULL,

    matricula VARCHAR(80) NOT NULL UNIQUE, -- unique significa que a matricula é unica, não pode ser repetida

    numero VARCHAR(20) NOT NULL,

    email VARCHAR(80) NOT NULL,

    desconto\_func INT NOT NULL

);

CREATE TABLE vendas (

    id\_venda BIGINT PRIMARY KEY AUTO\_INCREMENT,

    funcionario VARCHAR(80) NOT NULL,

    valor\_venda DOUBLE(10,2) NOT NULL,

    desconto\_venda DOUBLE(10,2)

);

-- Trigger para inserção

DELIMITER //

CREATE TRIGGER inserir\_desconto

BEFORE INSERT ON vendas -- depois de inserir na tabela vendas

FOR EACH ROW

BEGIN

    IF NEW.desconto\_venda IS NULL THEN -- se desconto\_venda for null

        SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01); -- mudar desconto\_venda baseado na variável alterada pelo backend

    END IF;

END //

DELIMITER ;

-- Trigger para atualização

DELIMITER //

CREATE TRIGGER atualizar\_desconto -- mesma coisa só que para atualização

BEFORE UPDATE ON vendas

FOR EACH ROW

BEGIN

    IF NEW.desconto\_venda IS NULL THEN

        SET NEW.desconto\_venda = NEW.valor\_venda \* (@porcentagemDesconto \* 0.01);

    END IF;

END //

DELIMITER ;

-- alteração de nome e desconto baseado na tabela funcionarios

DELIMITER //

CREATE TRIGGER atualizar\_funcionario\_vendas

BEFORE UPDATE ON funcionarios -- depois de atualizar a tabela funcionarios

FOR EACH ROW

BEGIN

    UPDATE vendas

    SET funcionario = NEW.nome\_func, -- mudar funcionario, baseado na alteração de funcionarios

        desconto\_venda = vendas.valor\_venda \* (NEW.desconto\_func \* 0.01) -- mudar desconto\_venda baseado na alteração de desconto\_func

    WHERE funcionario = OLD.nome\_func; -- com o parametro do mome que foi alterado, por isso OLD

END //

DELIMITER ;