



UNIVERSIDADE
Estácio de Sá

Universidade	Estácio de Sá
Campus	Polo de Cobilândia / Vila – Velha/ES
Nome do Curso	Desenvolvimento Full Stack
Nome da Disciplina	RPG0017 - Vamos integrar sistemas
Turma	9001
Semestre	Primeiro Semestre de 2024
Integrantes do Grupo	Tiago de Jesus Pereira Furtado
Matrícula	202306189045

**VILA VELHA
2024**

Implementação de sistema cadastral com interface Web, baseado nas tecnologias de Servlets, JPA e JEE.

👉 1º Procedimento | Camadas de Persistência e Controle

1- Objetivo da Prática:

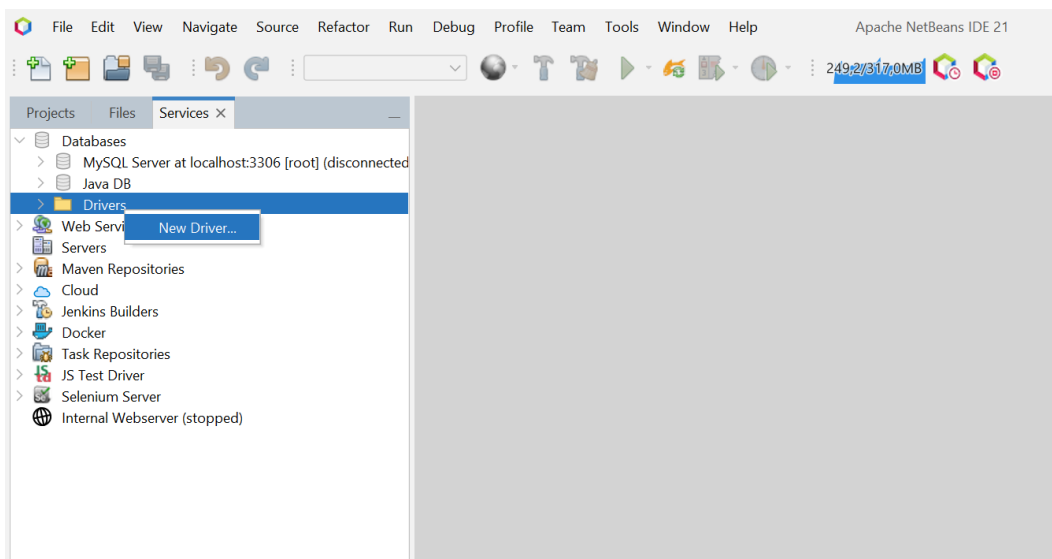
- Implementar persistência com base em JPA.
- Implementar regras de negócio na plataforma JEE, através de EJBs.
- Implementar sistema cadastral Web com base em Servlets e JSPs.
- Utilizar a biblioteca Bootstrap para melhoria do design.
- No final do exercício, o aluno terá criado todos os elementos necessários para exibição e entrada de dados na plataforma Java Web, tornando-se capacitado para lidar com contextos reais de aplicação.



3 - Todos os códigos solicitados neste roteiro de aula:

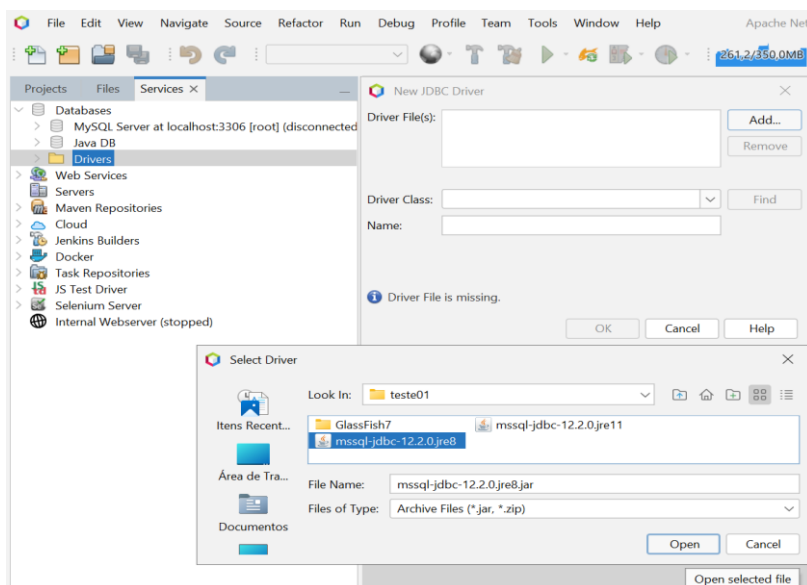
❖ Configurar a conexão com SQL Server via NetBeans e o pool de conexões no GlassFish Server 7.0.1:

- ✓ Na aba de **Serviços**, divisão **Banco de Dados**, clique com o botão
- ✓ direito em **Drivers** e escolha **Novo Driver**.

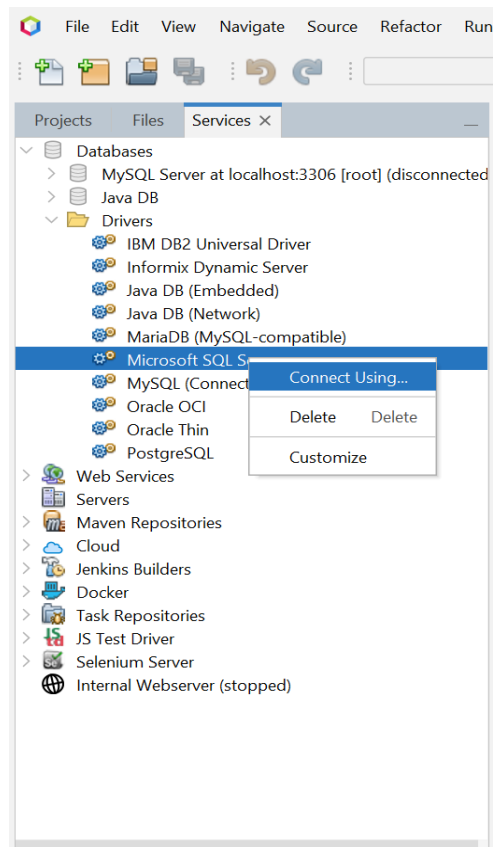
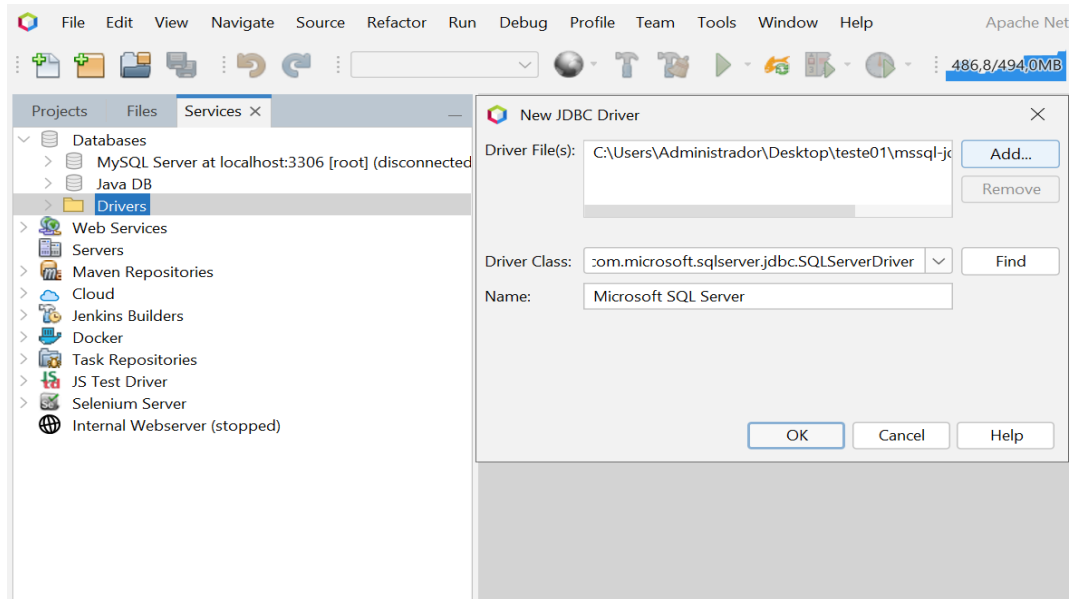


“Descrição”

Na janela que se abrirá, clicar em **Add** (Adicionar), selecionar o arquivo **mssql-jdbc-12.2.0.jre8.jar**, que é parte do arquivo **zip** encontrado no endereço seguinte, e finalizar com **Ok**



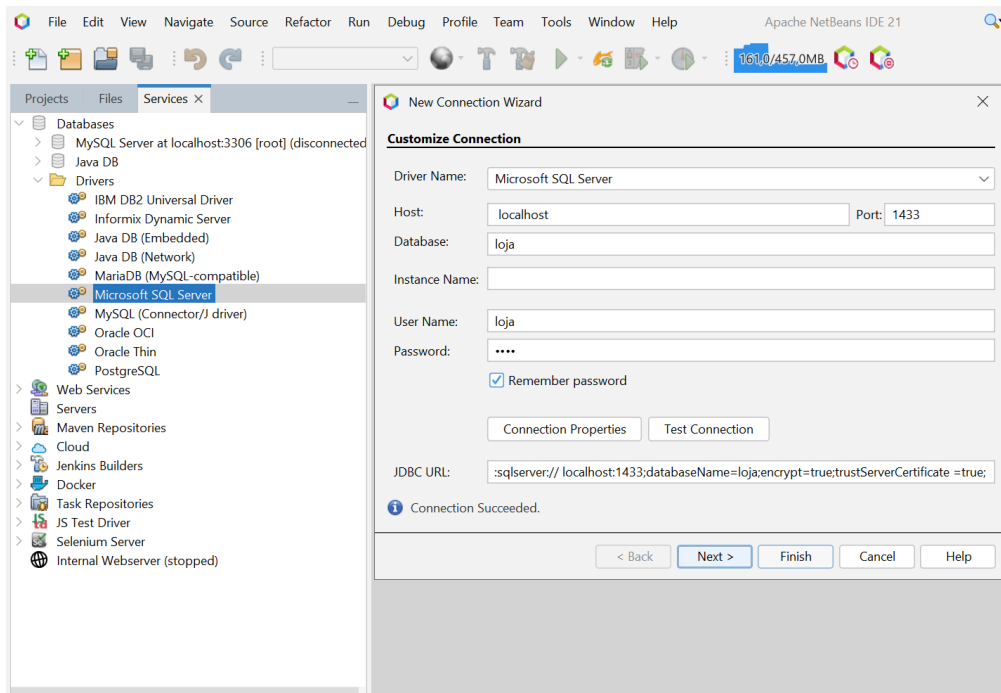
- O reconhecimento será automático, e podemos definir uma conexão com o clique do botão direito sobre o driver e escolha de **Conectar Utilizando**.



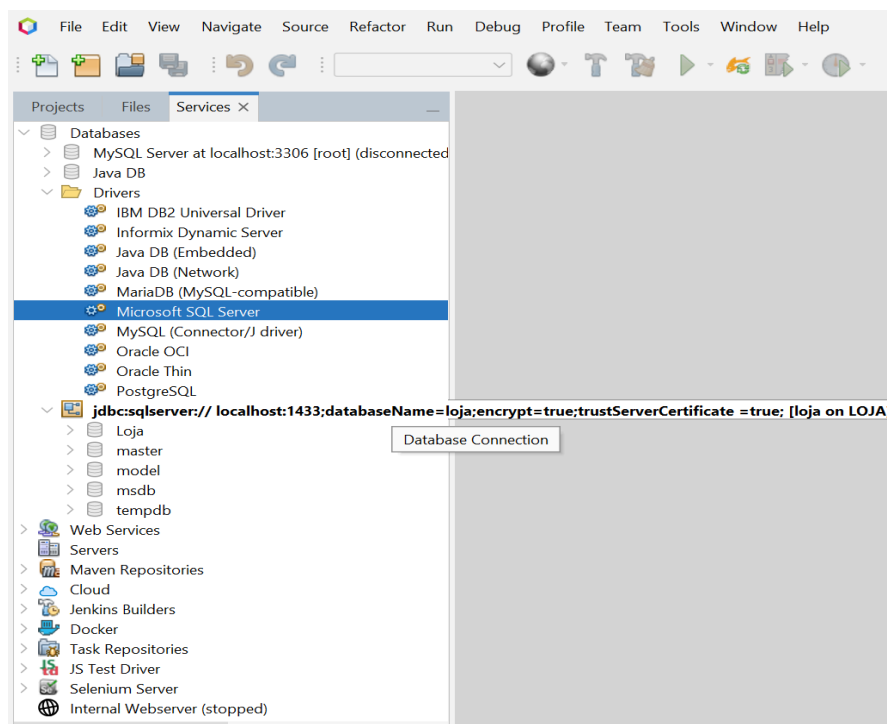
- ✓ Para os campos **database**, **user** e **password**, utilizar o valor **loja**, de acordo com os elementos criados em exercício anterior sobre a criação do banco de dados de exemplo, marcando também a opção **Lembrar Senha**.

Para o campo JDBC URL deve ser utilizada a seguinte expressão:

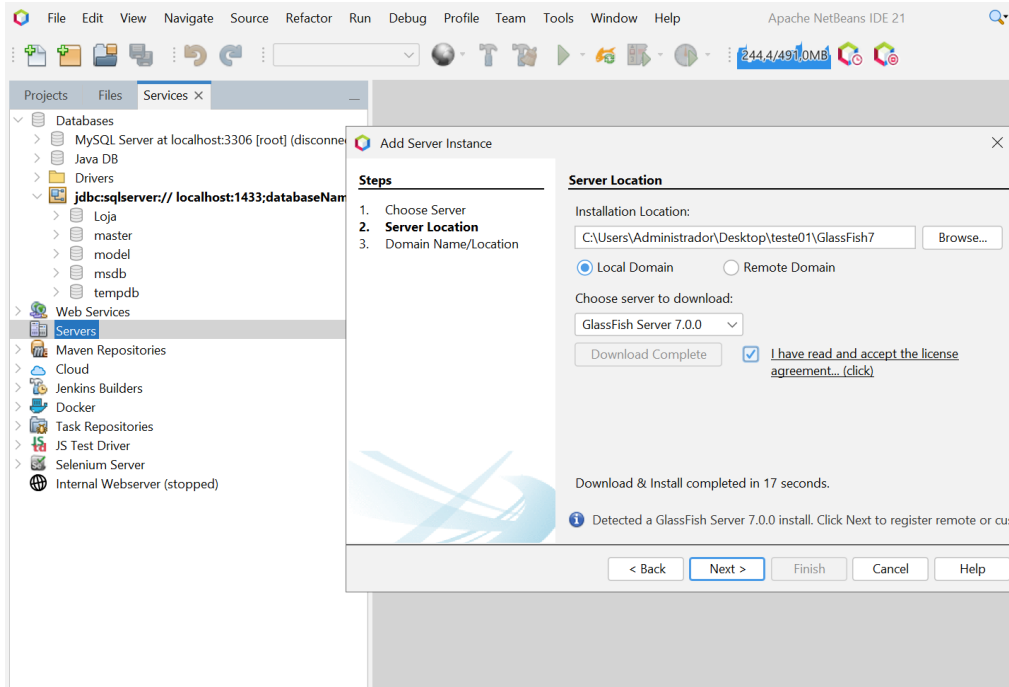
```
jdbc:sqlserver://  
localhost:1433; dbName=loja; encrypt=true; trustServerCertificate  
=true;
```



- ✓ Clicar em Testar Conexão e, estando tudo certo, Finalizar:



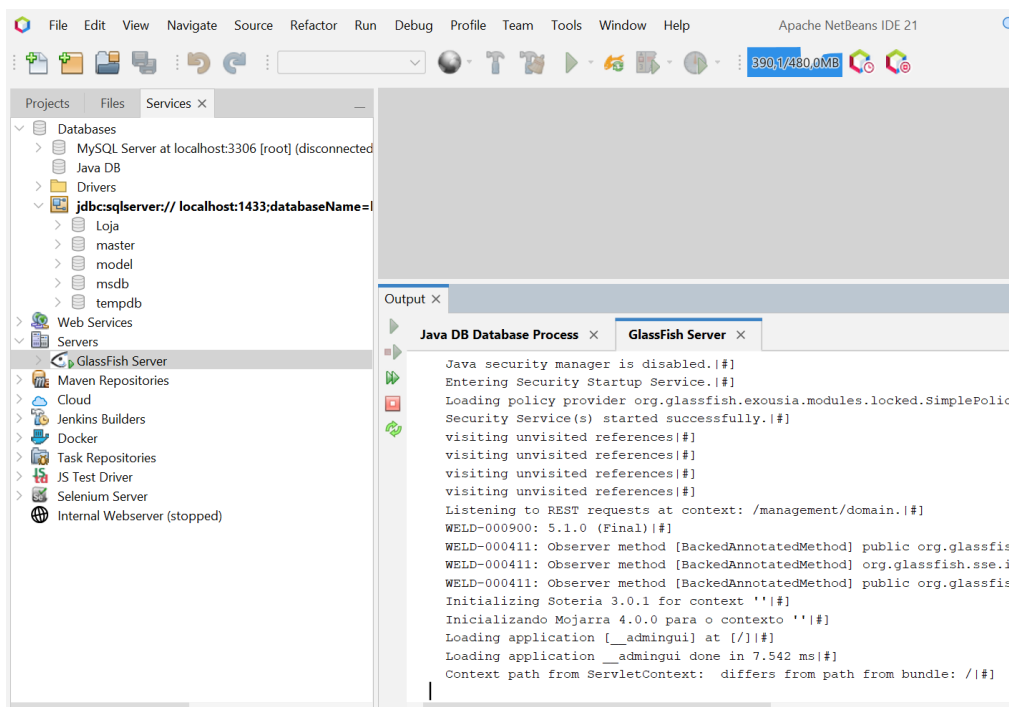
- ✓ Na divisão **Servidores**, verificar se o GlassFish 7.0.0 (ou posterior) está instalado, e caso não esteja, adicionar o servidor, via clique com o botão direito e escolha da opção **Add Server**, efetuando o download a partir da própria janela que se abrirá.



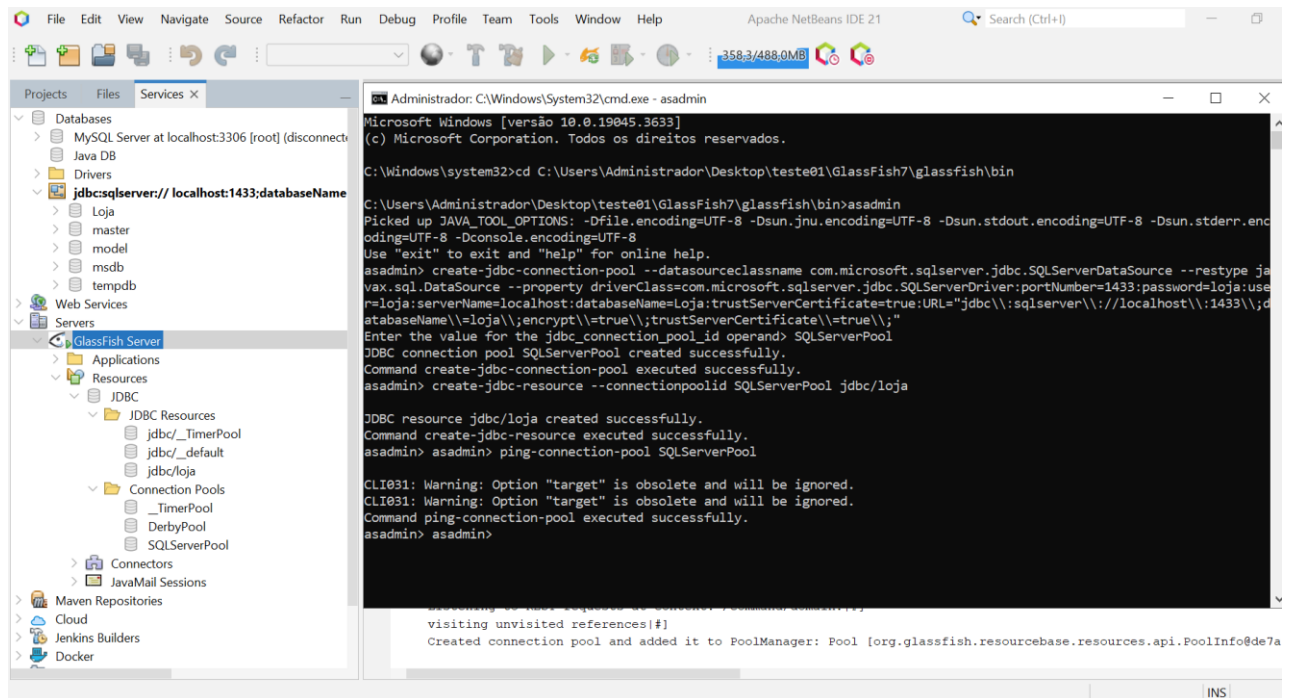
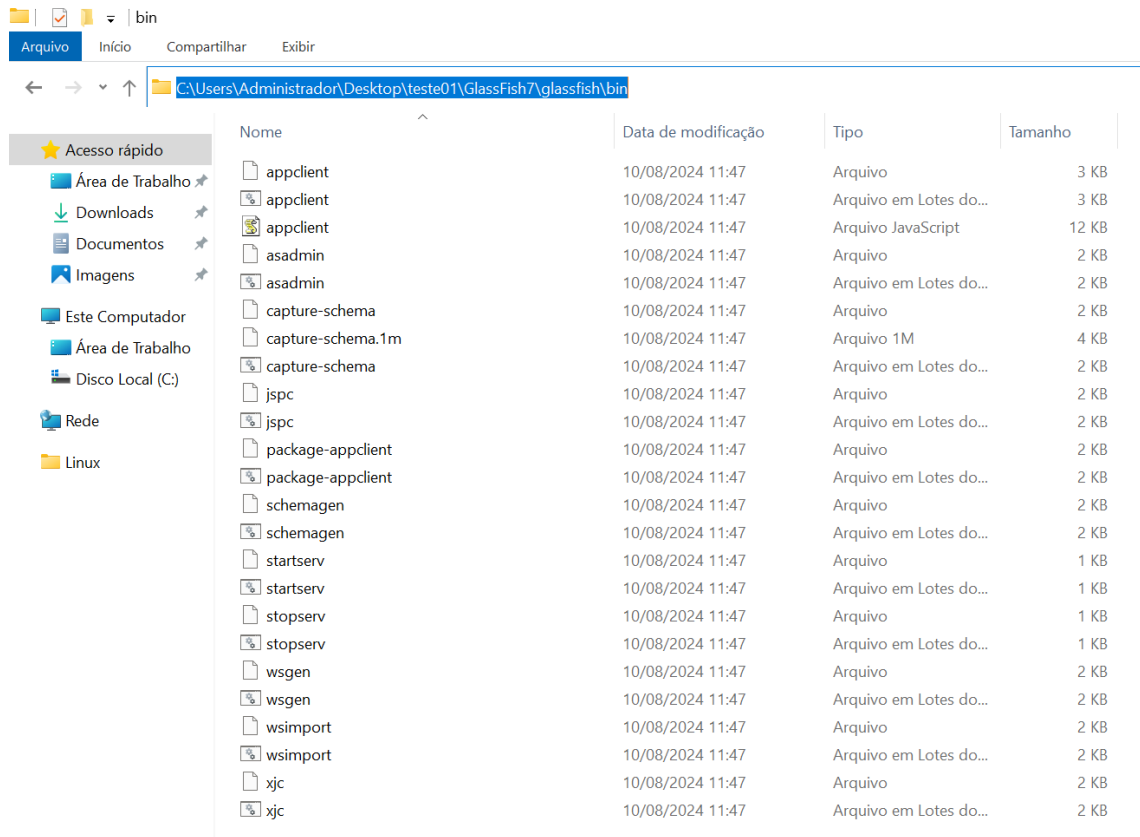
“Descrição”

Adicionando o Servidor GlassFish 7.0.0, Obs.: Foram feitas algumas mudanças nos programas que foram solicitados no Trabalho como foi passado a versão que pede no trabalho e a 6.2.1 porem foi utilizado versão mais recentes.

❖ Iniciar o servidor GlassFish a partir do NetBeans.

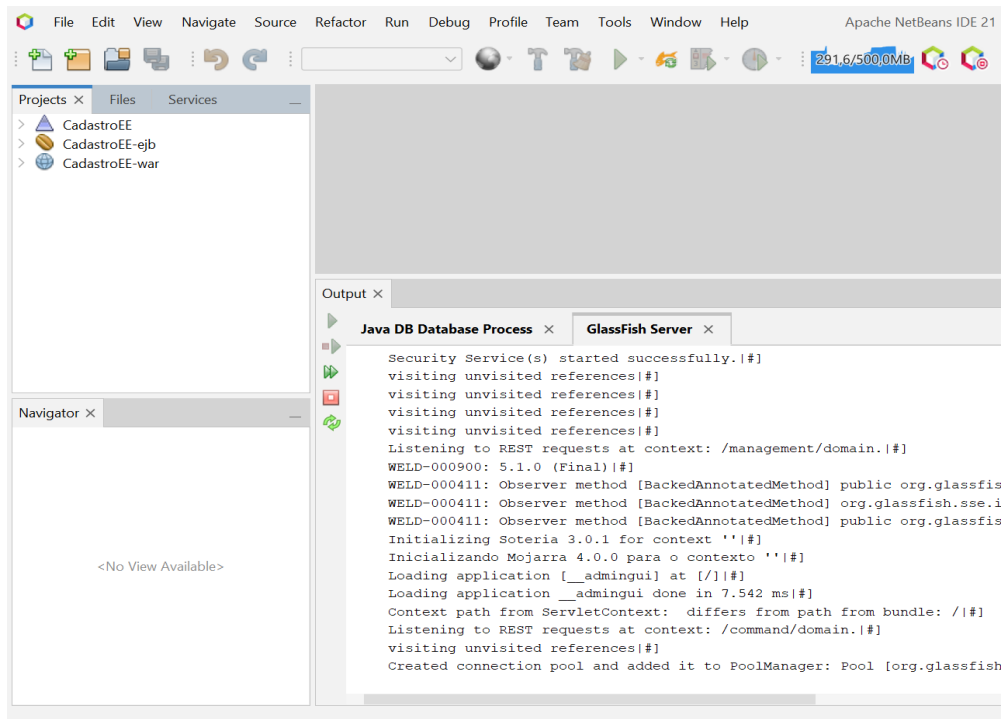
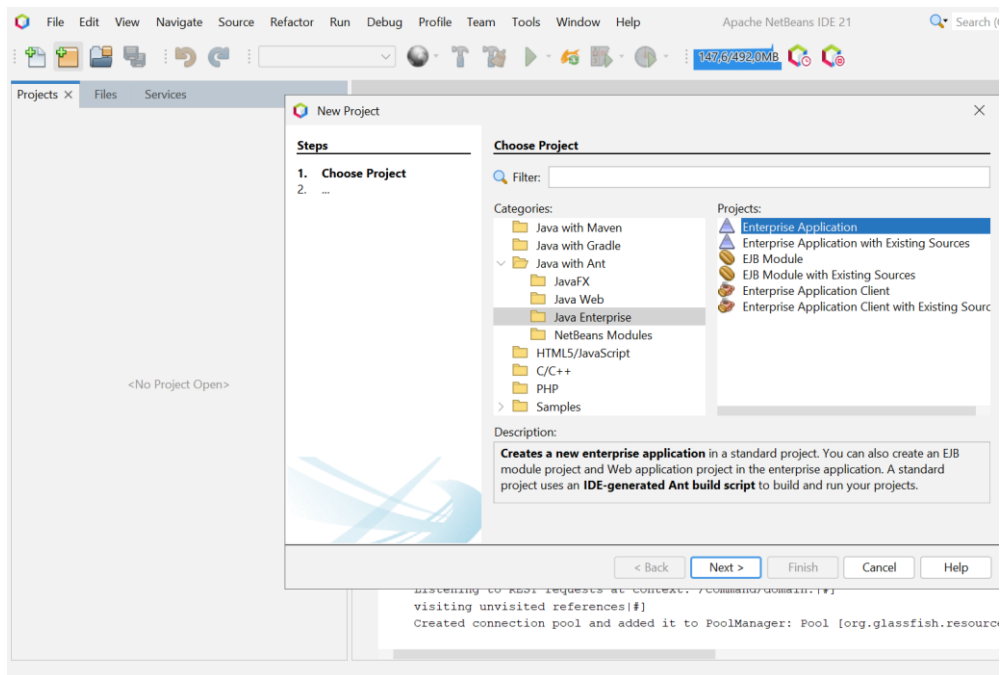


- ✓ Através da linha de comando, executar o comando **asadmin**, no diretório **bin** do **GlassFish**.
- ✓ No **prompt do asadmin**, executar o comando apresentado a seguir:



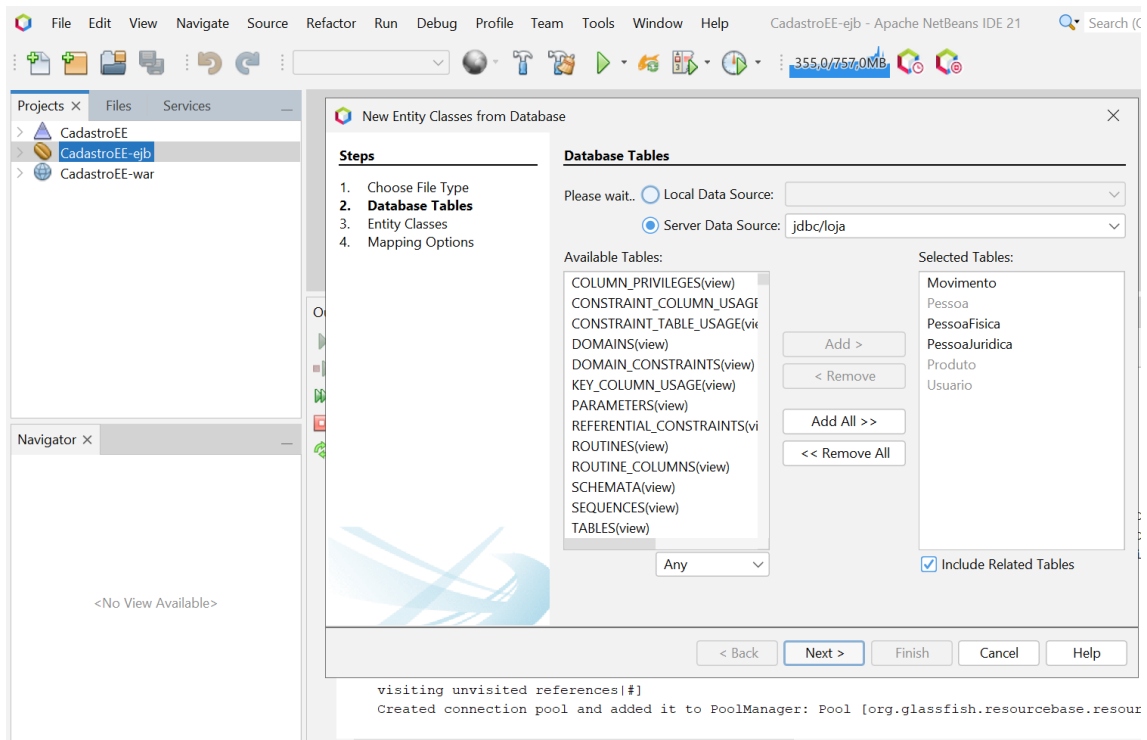
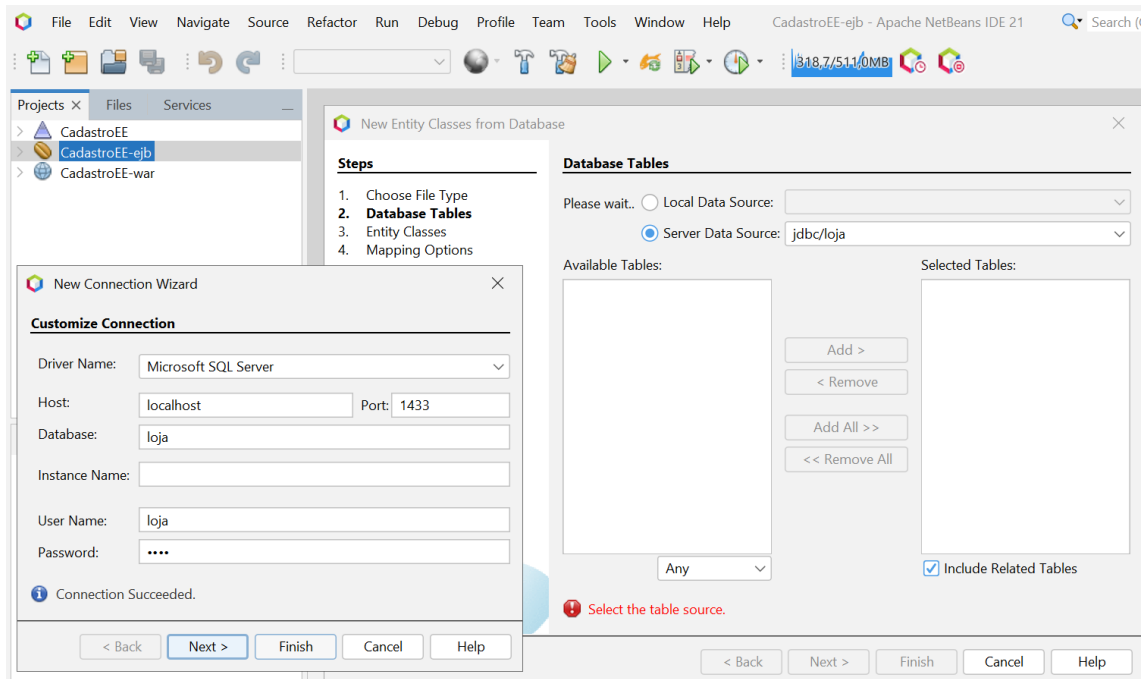
❖ Criar o aplicativo corporativo no NetBeans:

- ✓ Criar um projeto do tipo **Ant...Java Enterprise...Enterprise Application**.
- ✓ Adotar o nome **CadastroEE**, com escolha do servidor **GlassFish**, além de plataforma **Jakarta JEE 10**.
- ✓ Serão gerados três projetos, onde o principal encapsula o arquivo **EAR**, tendo os outros dois, **CadastroEE-ejb** e **CadastroEE-war**, como projetos dependentes, relacionados aos elementos **JPA**, **JEE** e **Web**.

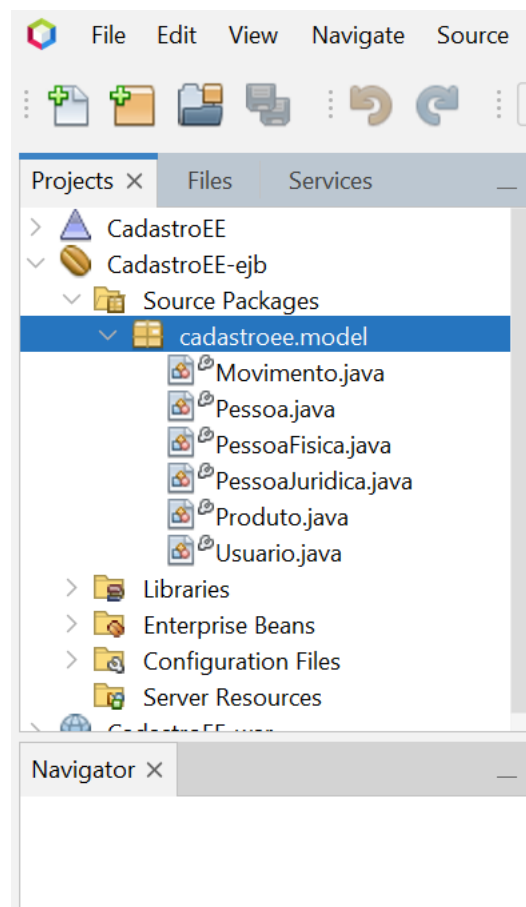
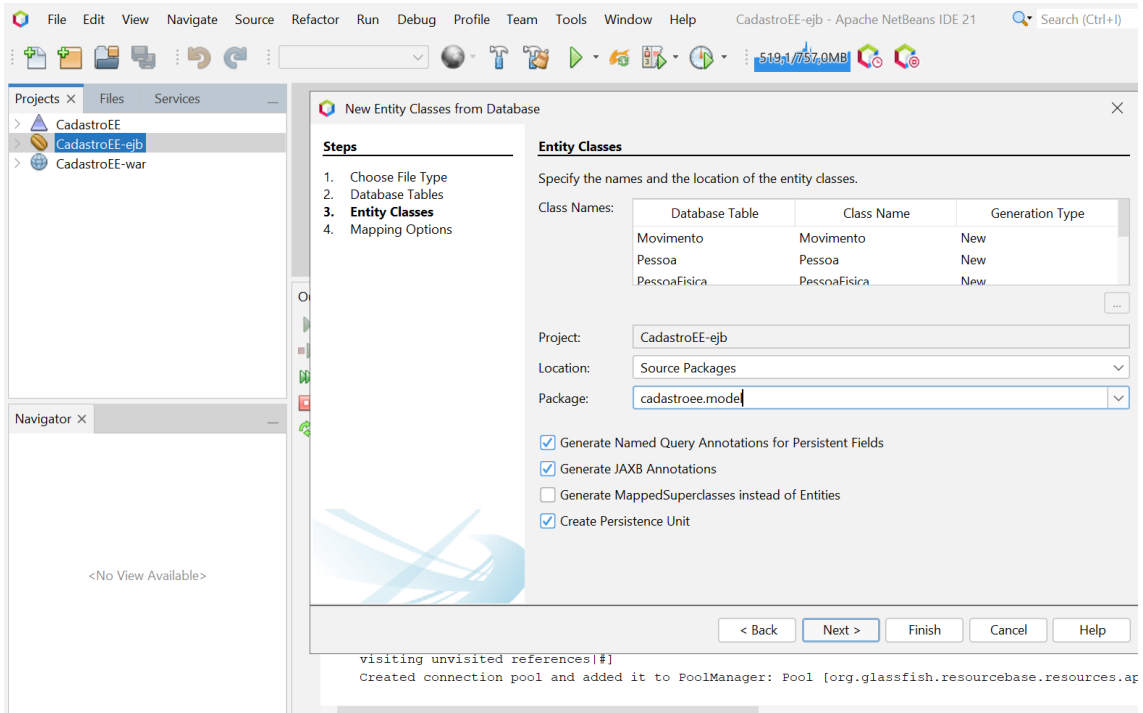


➤ **Definir as camadas de persistência e controle no projeto CadastroEE-ejb.**

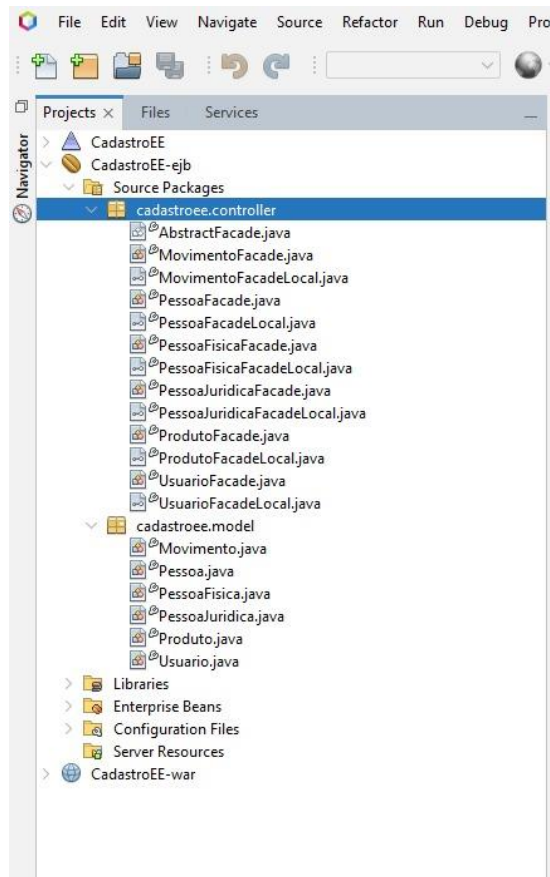
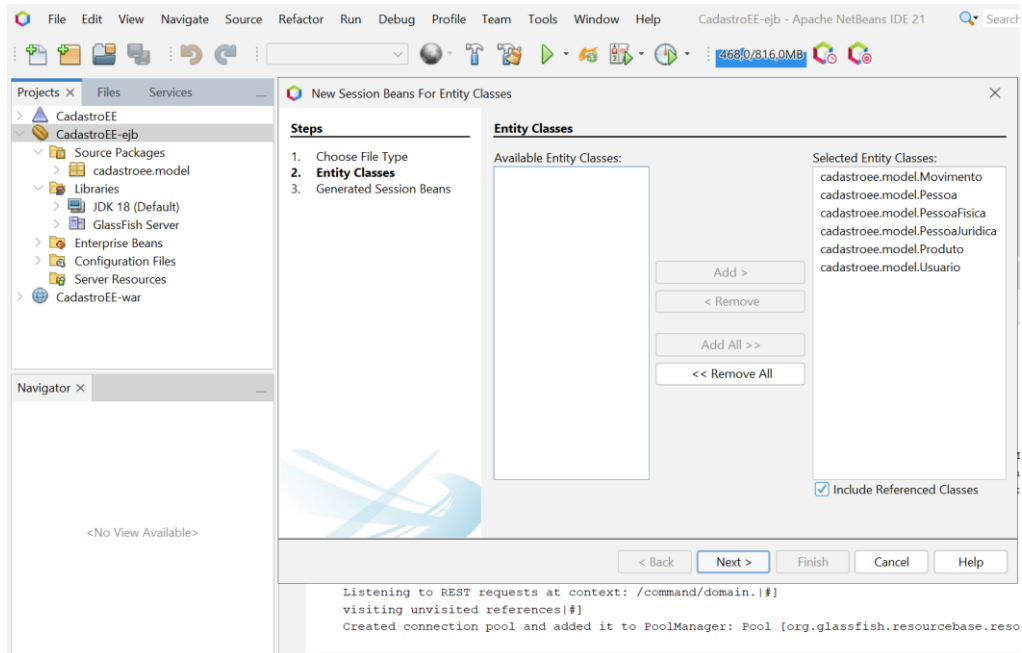
- ✓ Criar as entidades JPA através de New Entity Classes from Database.
- ✓ Selecionar jdbc/loja como Data Source, e selecionar todas as tabelas.



- ✓ No passo seguinte, definir o pacote como **cadastroee.model**, além de marcar a opção para criação do arquivo **persistence.xml**.

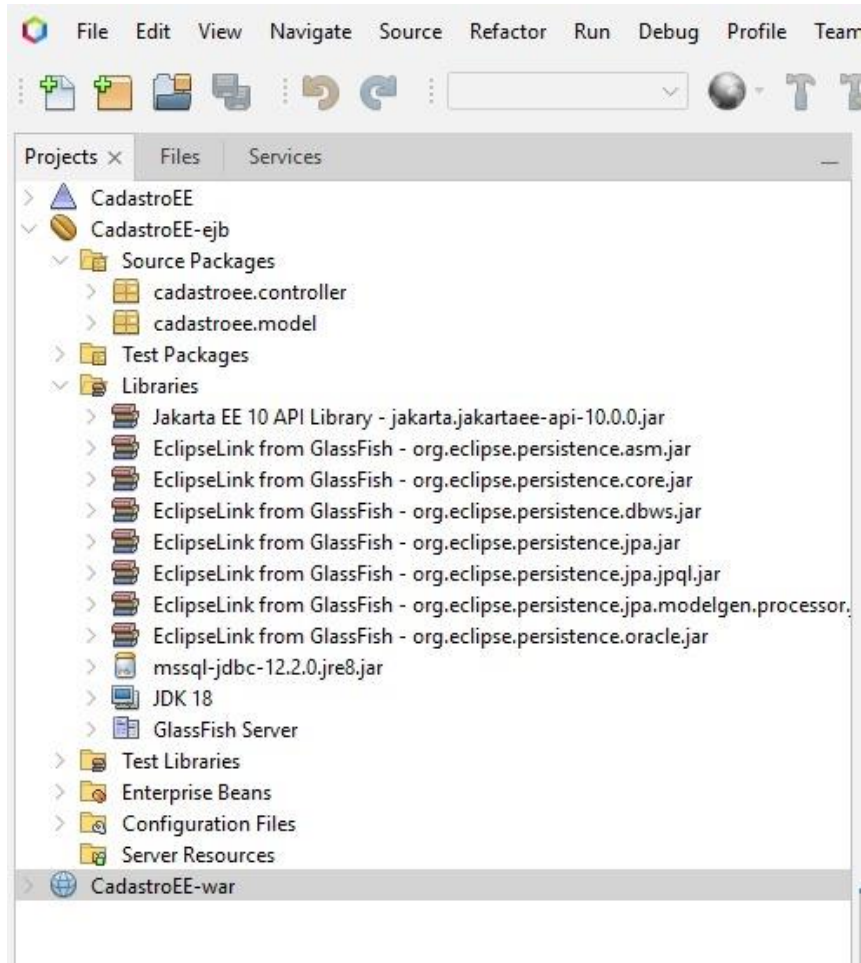


- ✓ Em seguida, adicionar os componentes EJB ao projeto, através da opção **New Session Beans for Entity Classes**.
- ✓ Selecionar **todas as entidades**, marcar a geração da **interface local**, além de definir o nome do pacote como **cadastroee.controller**.
- ✓ Serão gerados todos os **Session Beans**, com o sufixo **Facade**, bem como as **interfaces**, com o sufixo **FacadeLocal**.



❖ Efetuar pequenos acertos no projeto, para uso do Jakarta:

- ✓ Adicionar a biblioteca **Jakarta EE 10 API** ao projeto **CadatroEE-ejb**.
- ✓ Criados os componentes e ajustadas as bibliotecas, o projeto deverá ficar como apresentado a seguir.
- ✓ Modificar **TODAS** as importações de pacotes **javax** para **jakarta**, em todos os arquivos do projeto **CadastroEE-ejb**.

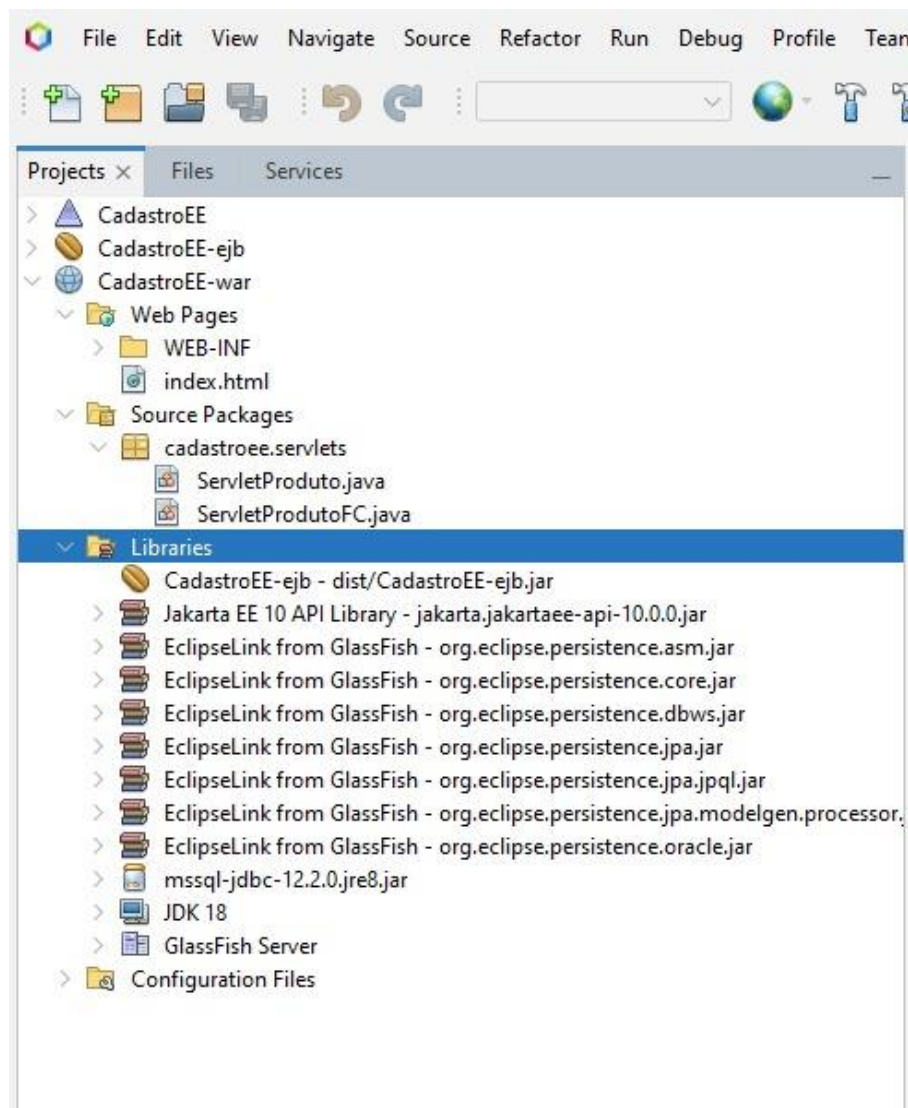


```
package cadastroee.model;
```

```
import jakarta.persistence.Basic;  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.persistence.NamedQueries;  
import jakarta.persistence.NamedQuery;  
import jakarta.persistence.OneToOne;  
import jakarta.persistence.Table;  
import jakarta.validation.constraints.NotNull;  
import jakarta.validation.constraints.Size;  
import jakarta.xml.bind.annotation.XmlRootElement;  
import jakarta.xml.bind.annotation.XmlTransient;  
import java.io.Serializable;
```

❖ Criar um Servlet de teste no projeto CadastroEE-war

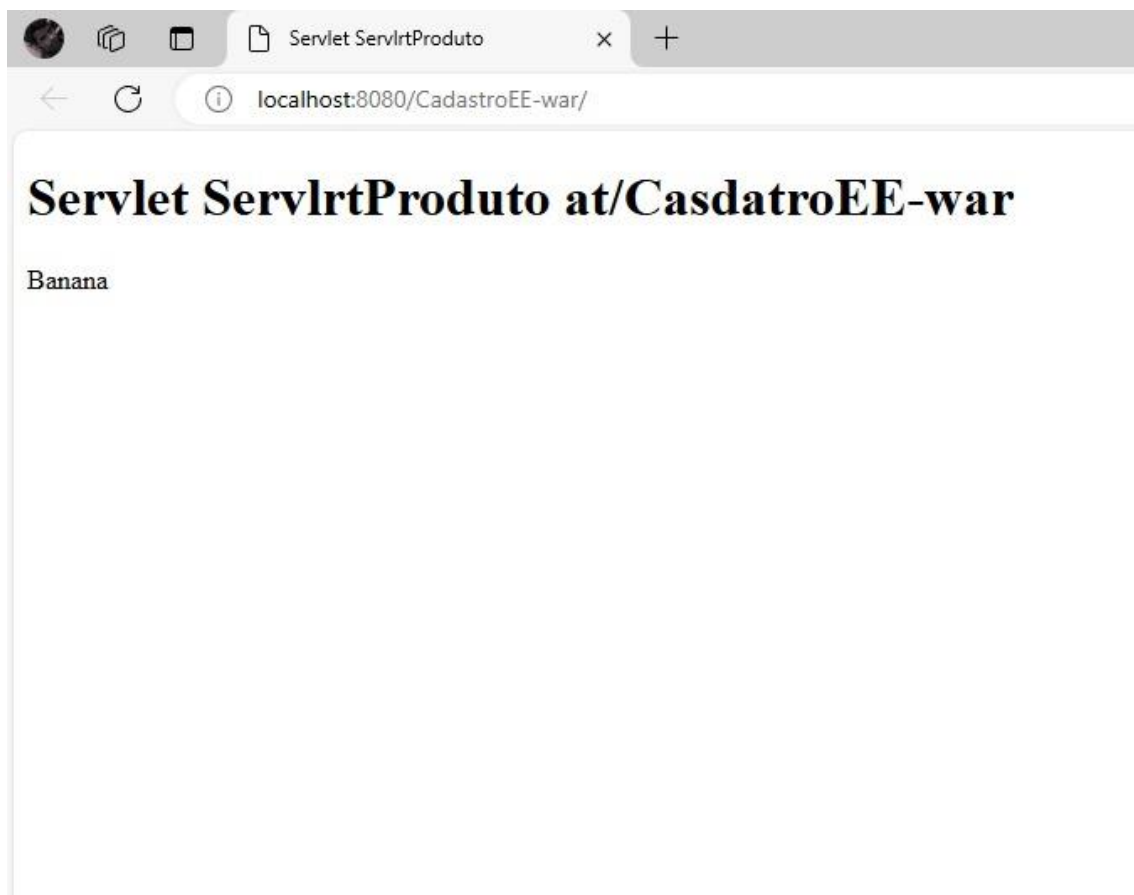
- ✓ Utilizar o clique do botão direito e escolha da opção **New...Servlet**
- ✓ Definir o nome do Servlet como **ServletProduto**, e nome do pacote como **cadastroee.servlets**
- ✓ Adicionar, no código do Servlet, a referência para a interface do EJB **@EJB ProdutoFacadeLocal Facade;**
- ✓ Modificar a resposta do Servlet, utilizando o **Facade** para recuperar os dados e apresentá-los na forma de lista HTML
- ✓ Efetuar novos acertos no projeto, para uso do Jakarta:
Adicionar a biblioteca **Jakarta EE Web 10 API** ao projeto **CadatroEE-war**
- ✓ Criado o Servlet e ajustadas as bibliotecas, o projeto deverá ficar como apresentado a seguir:



4 - Os resultados da execução dos códigos também devem ser apresentados:

❖ Executar o projeto:

- ✓ A execução deve ser efetuar com o uso de **Run** ou **Deploy** no projeto
- ✓ principal (**CadastroEE**), simbolizado por um triângulo
- ✓ Acessar o endereço a seguir, para testar o Servlet **http://**
- ✓ **localhost:8080/CadastroEE-war/ServletProduto**
- ✓ Tendo alimentado a base via SQL Server Management Studio, ou pela
- ✓ aba de serviços do NetBeans, deve ser obtida uma saída como a seguinte:



5 – Análise e Conclusão:

❖ Como é organizado um projeto corporativo no NetBeans?

❖ Organizar um projeto corporativo no NetBeans envolve os seguintes passos principais:

- ✓ **Criação do Projeto:** Inicie um novo projeto selecionando o tipo (Java SE, Web, Java EE, etc.) e defina o nome e a localização.
- ✓ **Configuração da Estrutura:** Organize o código em módulos, configure dependências (via Maven ou Gradle), e siga a estrutura de pastas padrão (src, resources, etc.).
- ✓ **Configuração do Ambiente:** Configure o servidor de aplicações (GlassFish, Tomcat, etc.), e as conexões com bases de dados via JDBC.
- ✓ **Compilação e Deploy:** Configure o processo de build, testes automatizados e deployment, utilizando as ferramentas integradas do NetBeans.
- ✓ **Gestão do Código-Fonte:** Use Git para versionamento, integração contínua (CI) e colaboração em equipe.

❖ Conclusão:

- ✓ Organizar um projeto corporativo no NetBeans requer uma abordagem sistemática para garantir que o código esteja bem estruturado, fácil de manter e escalável. O NetBeans oferece uma série de ferramentas e integrações que facilitam este processo.

❖ Qual o papel das tecnologias JPA e EJB na construção de um aplicativo para a plataforma Web no ambiente Java?

❖ No ambiente Java para a construção de aplicações Web, as tecnologias JPA e EJB desempenham papéis complementares:

- ✓ **JPA (Java Persistence API):** Facilita o mapeamento objeto-relacional, permitindo que objetos Java sejam persistidos em bases de dados relacionais de forma transparente, simplificando o acesso e gestão de dados.
- ✓ **EJB (Enterprise JavaBeans):** Oferece uma arquitetura para construir componentes de negócio que suportam transações, segurança e escalabilidade, permitindo o desenvolvimento de aplicações corporativas robustas e distribuídas.

❖ Conclusão:

- ✓ Essas tecnologias juntas proporcionam uma base sólida para a construção de aplicações empresariais complexas e escaláveis na plataforma Java.

❖ Como o NetBeans viabiliza a melhoria de produtividade ao lidar com as tecnologias JPA e EJB?

❖ O NetBeans melhora a produtividade ao lidar com JPA e EJB de várias maneiras:

✓ **Assistentes e Geradores de Código:**

- O NetBeans oferece assistentes que geram automaticamente o código básico para entidades JPA e EJBs. Por exemplo, você pode gerar classes de entidades a partir de uma base de dados existente, poupando tempo na codificação manual.

✓ **Integração com Servidores de Aplicação:**

- A integração direta com servidores de aplicação (como GlassFish e Payara) facilita a criação, o deployment e o teste de EJBs em ambiente real, acelerando o ciclo de desenvolvimento.

✓ **Editor com Suporte a Anotações:**

- O editor do NetBeans oferece suporte avançado a anotações JPA e EJB, com autocompletar, dicas de código e detecção de erros em tempo real, o que reduz erros e aumenta a eficiência.

✓ **Ferramentas de Persistência:**

- O NetBeans inclui ferramentas para mapeamento objeto-relacional e geração de consultas JPQL, ajudando a visualizar e manipular os dados persistentemente sem sair do IDE.

✓ **Debugging e Perfilamento:**

- Ferramentas de depuração integradas permitem a inspeção de transações, ciclos de vida de EJBs e operações JPA, facilitando a identificação e correção de problemas de forma mais rápida.

❖ **Conclusão:**

- ✓ Esses recursos permitem que os desenvolvedores foquem na lógica de negócios e na persistência de dados, reduzindo o tempo necessário para tarefas repetitivas e propensas a erros.

❖ O que são Servlets, e como o NetBeans oferece suporte à construção desse tipo de componentes em um projeto Web?

Servlets são componentes Java que atuam como controladores na arquitetura de aplicações Web. Eles processam requisições HTTP do cliente (como navegadores) e geram respostas, geralmente na forma de HTML, JSON ou XML. Servlets são essenciais para implementar a lógica de negócios e a interação com o cliente em aplicações Web.

Suporte do NetBeans à Construção de Servlets

❖ O NetBeans oferece várias funcionalidades que facilitam o desenvolvimento de Servlets:

✓ **Criação Automática de Servlets:**

- O NetBeans possui assistentes que permitem a criação rápida de Servlets. Você pode configurar o nome, pacote e parâmetros de inicialização diretamente na interface gráfica, e o NetBeans gera o código esqueleto automaticamente.

✓ **Editor de Código:**

- O editor do NetBeans oferece autocompletar, realce de sintaxe e sugestões de código enquanto você desenvolve o Servlet, o que acelera a codificação e ajuda a evitar erros.

✓ **Configuração Simplificada:**

- O NetBeans facilita a configuração do web.xml ou o uso de anotações como @WebServlet para mapear URLs ao Servlet, gerando e organizando essas configurações automaticamente.

✓ **Deploy e Teste:**

- Com integração direta aos servidores de aplicação (como Tomcat, GlassFish, etc.), o NetBeans permite que você implemente e teste Servlets com um clique, tornando o ciclo de desenvolvimento mais rápido e eficiente.

✓ **Depuração e Monitoramento:**

- Ferramentas de depuração integradas no NetBeans permitem monitorar o comportamento do Servlet durante a execução, permitindo a inspeção de variáveis, acompanhamento de fluxos e resolução de problemas em tempo real.

❖ **Conclusão:**

- ✓ Esses recursos ajudam os desenvolvedores a criar, configurar, testar e depurar Servlets de maneira eficiente, melhorando a produtividade e a qualidade do desenvolvimento Web.

❖ **Como é feita a comunicação entre os Servlets e os Session Beans do pool de EJBs?**

- ✓ A comunicação entre **Servlets** e **Session Beans** no pool de EJBs em uma aplicação Java EE é feita principalmente através de **Injeção de Dependências** e **lookup**. Aqui está como isso ocorre:

❖ **Injeção de Dependências (@EJB)**

- ✓ A maneira mais comum de um Servlet se comunicar com um Session Bean é através da injeção de dependência usando a anotação **@EJB**.
 - ✓ No Servlet, você pode injetar um Session Bean diretamente como um atributo de classe. O container Java EE cuida de instanciar o bean e gerenciá-lo.
- ✓ Exemplo:

```
@WebServlet("/meuServlet")
public class MeuServlet extends HttpServlet {

    @EJB
    private MeuSessionBean meuSessionBean;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Usando o session bean injetado
        String resultado = meuSessionBean.minhaOperacao();
        response.getWriter().write(resultado);
    }
}
```

- ✓ Aqui, meuSessionBean é um Session Bean que é automaticamente injetado pelo container e pode ser usado diretamente no Servlet.

❖ Lookup via JNDI

- ✓ Outra maneira de um Servlet acessar um Session Bean é através de um **lookup** via JNDI (Java Naming and Directory Interface). Embora menos comum em comparação com a injeção, o lookup JNDI é útil em cenários onde o nome do bean ou o ambiente é dinâmico.
- ✓ Exemplo:

```
@WebServlet("/meuServlet")
public class MeuServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        try {
            InitialContext ctx = new InitialContext();
            MeuSessionBean meuSessionBean = (MeuSessionBean) ctx.lookup("java:module/MeuSessionBean");
            String resultado = meuSessionBean.minhaOperacao();
            response.getWriter().write(resultado);
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
}
```

- ✓ Aqui, o Session Bean é procurado pelo seu nome JNDI e, em seguida, invocado no Servlet.

❖ Passagem de Dados e Controle de Transações

- ✓ **Passagem de Dados:** Dados podem ser passados do Servlet para o Session Bean como parâmetros de método, e o resultado pode ser devolvido ao Servlet para processamento adicional ou resposta ao cliente.
- ✓ **Controle de Transações:** Session Beans, especialmente Stateless e Stateful, podem gerenciar transações automaticamente. O Servlet pode chamar métodos do Session Bean, que executa operações de negócios e manipulação de dados de forma transacional.

❖ Ciclo de Vida e Escopo

- ✓ **Stateless Session Beans:** Geralmente usados para operações rápidas e sem estado, são ideais para ser chamados por Servlets, pois o container pode gerenciar seu ciclo de vida eficientemente.
- ✓ **Stateful Session Beans:** Mantêm estado entre chamadas e podem ser utilizados em cenários onde o estado do cliente precisa ser preservado ao longo de múltiplas interações.

❖ Conclusão:

- ✓ Essa interação entre Servlets e Session Beans permite que a lógica de apresentação e de negócios sejam separadas, promovendo uma arquitetura de software mais modular e fácil de manter.

👉 2º Procedimento | Interface Cadastral com Servlet e JSPs

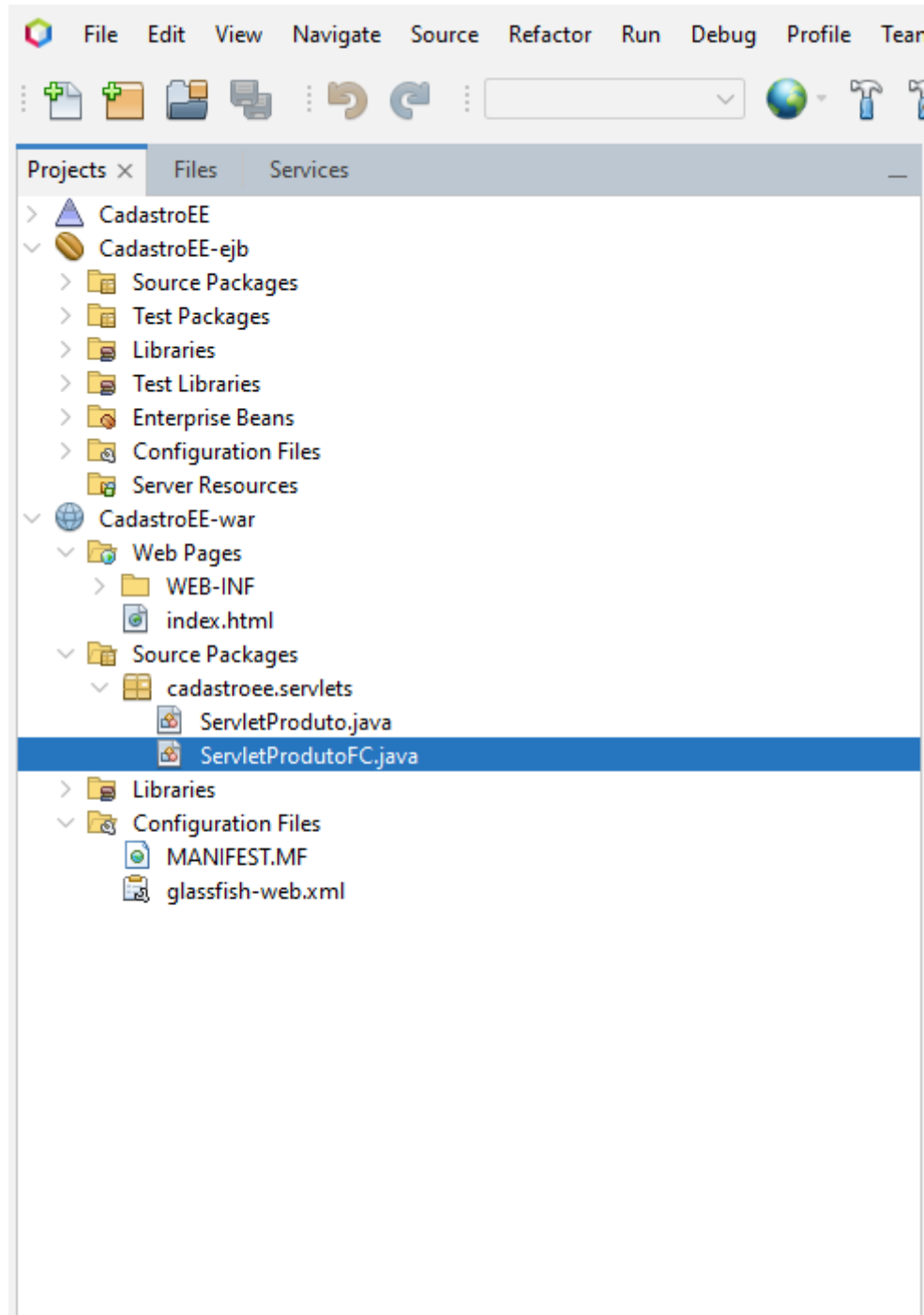
Objetivo da Prática:

- Implementar persistência com base em JPA.
- Implementar regras de negócio na plataforma JEE, através de EJBs.
- Implementar sistema cadastral Web com base em Servlets e JSPs.
- Utilizar a biblioteca Bootstrap para melhoria do design.
- No final do exercício, o aluno terá criado todos os elementos necessários para exibição e entrada de dados na plataforma Java Web, tornando-se capacitado para lidar com contextos reais de aplicação.

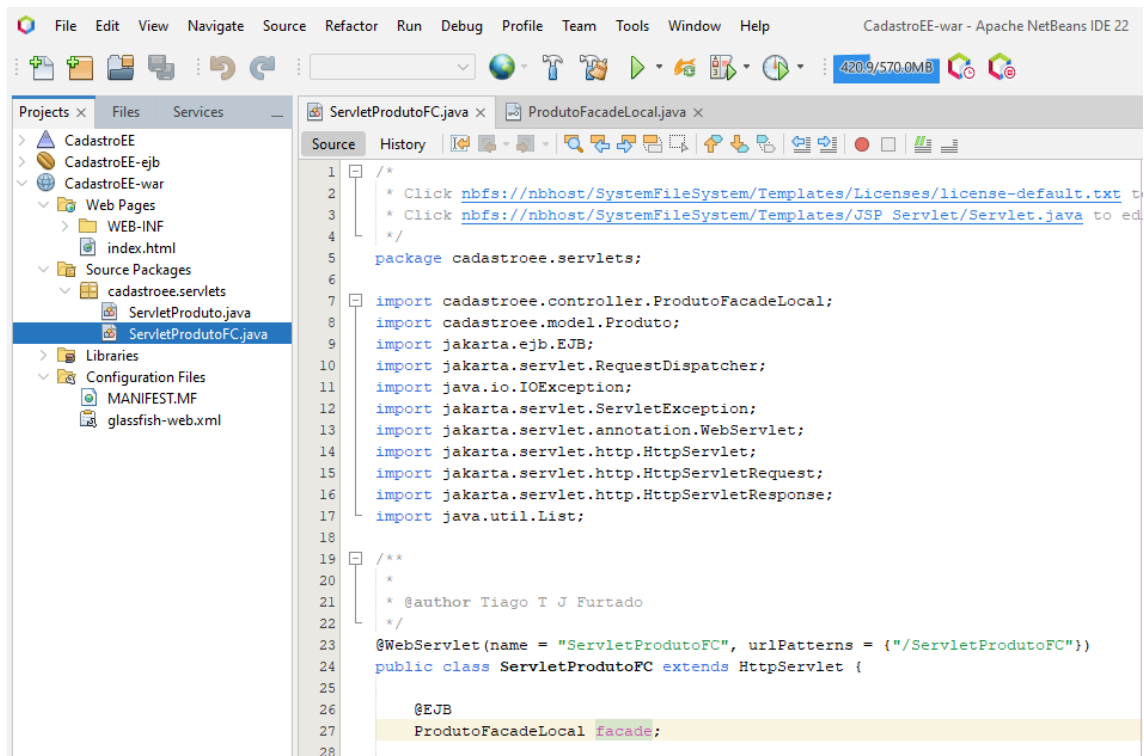


Todos os códigos solicitados neste roteiro de aula:

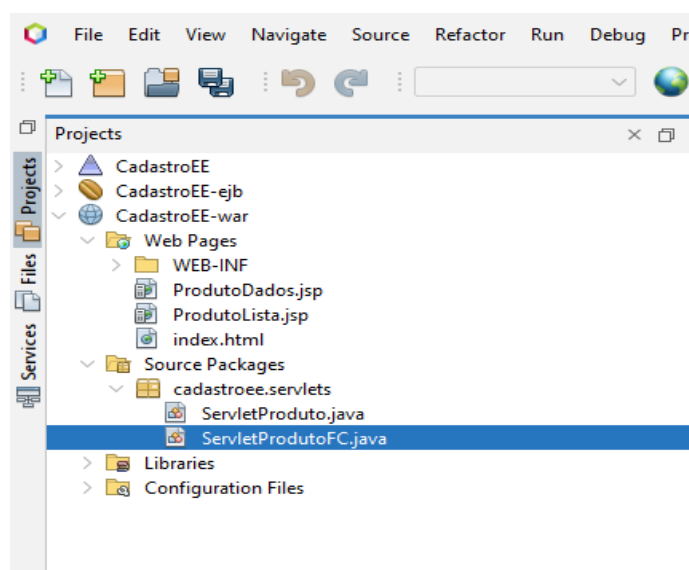
- ✓ Criar um Servlet com o nome ServletProdutoFC, no projeto CadastroEE war:



- ✓ Utilizar o padrão Front Controller
- ✓ Adicionar uma referência para ProdutoFacadeLocal, utilizando o nome facade para o atributo



- ✓ Apagar o conteúdo interno do método **processRequest**, e efetuar nele as modificações seguintes.
- ✓ Capturar o parâmetro **acao** a partir do **request**, o qual poderá assumir os valores **listar**, **incluir**, **alterar**, **excluir**, **formIncluir** e **formAlterar**.
- ✓ Definir a variável **destino**, contendo o nome do JSP de apresentação, que terá os valores **ProdutoDados.jsp**, para **acao** valendo **formAlterar** ou **formIncluir**, ou **ProdutoLista.jsp**, para as demais opções.
- ✓ Para o valor **listar**, adicionar a **listagem de produtos** como atributo da requisição (**request**), com a consulta efetuada via facade.
- ✓ Para o valor **formAlterar**, capturar o **id** fornecido como parâmetro do request, consultar a **entidade** via facade, e adicioná-la como atributo da requisição (**request**).
- ✓ Para o valor **excluir**, capturar o **id** fornecido como parâmetro do request, remover a entidade através do facade, e adicionar a **listagem de produtos** como atributo da requisição (**request**).
- ✓ Para o valor **alterar**, capturar o **id** fornecido como parâmetro do request, consultar a entidade através do facade, preencher os demais campos com os valores fornecidos no request, alterar os dados via facade e adicionar a **listagem de produtos** como atributo da requisição (**request**).
- ✓ Para o valor **incluir**, instanciar uma entidade do tipo Produto, preencher os campos com os valores fornecidos no request, inserir via facade e adicionar a **listagem de produtos** como atributo da requisição (**request**).
- ✓ Ao final redirecionar para **destino** via **RequestDispatcher**, obtido a partir do objeto request.



✓ Segue a Abaixo o Código das Referências acima:

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3   * Click nbfs://nbhost/SystemFileSystem/Templates/JSP_Servlet/Servlet.java to e
4   */
5   package cadastroee.serviets;
6
7   import cadastroee.controller.ProdutoFacadeLocal;
8   import cadastroee.model.Produto;
9   import jakarta.ejb.EJB;
10  import jakarta.servlet.RequestDispatcher;
11  import java.io.IOException;
12  import jakarta.servlet.ServletException;
13  import jakarta.servlet.annotation.WebServlet;
14  import jakarta.servlet.http.HttpServlet;
15  import jakarta.servlet.http.HttpServletRequest;
16  import jakarta.servlet.http.HttpServletResponse;
17  import java.util.List;
18
19  /**
20   *
21   * @author Tiago T J Furtado
22   */
23  @WebServlet(name = "ServletProdutoFC", urlPatterns = {"/ServletProdutoFC"})
24  public class ServletProdutoFC extends HttpServlet {
25
26      @EJB
27      ProdutoFacadeLocal facade;
28
29      @Override
30      protected void doGet(HttpServletRequest request, HttpServletResponse response)
31          throws ServletException, IOException {
32
33          String acao = request.getParameter("acao");
34          String destino = "";
35
36          switch (acao) {
37              case "formIncluir":
38                  destino = "ProdutoDados.jsp";
39                  break;
40
41              case "excluir":
42                  int idDel = Integer.parseInt(request.getParameter("id"));
43                  facade.remove(facade.find(idDel));
44                  List<Produto> delProdutos = facade.findAll();
45                  request.setAttribute("produtos", delProdutos);
46                  destino = "ProdutoLista.jsp";
47                  break;
48
49              case "formAlterar":
50                  int id = Integer.parseInt(request.getParameter("id"));
51                  Produto produto = facade.find(id);
52                  request.setAttribute("produto", produto);
53                  destino = "ProdutoDados.jsp";
54                  break;
55
56              default:
57                  List<Produto> produtos = facade.findAll();
58                  request.setAttribute("produtos", produtos);
59                  destino = "ProdutoLista.jsp";
60                  break;
61          }
62
63      RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
64      dispatcher.forward(request, response);
65  }
66
67      @Override
68      protected void doPost(HttpServletRequest request, HttpServletResponse response)
69          throws ServletException, IOException {
70
71          String acao = request.getParameter("acao");
72          acao = acao == null || acao.isEmpty() ? " " : acao;
73
74          String destino = "ProdutoLista.jsp";
75
76          switch (acao) {
77              case "incluir":
78                  int idProduto = Integer.parseInt(request.getParameter("idProduto"));
79                  String nome = request.getParameter("nome");
80                  int quantidade = Integer.parseInt(request.getParameter("quantidade"));
81                  Float precoVenda = Float.valueOf(request.getParameter("precoVenda"));
82
83                  Produto newProduto = new Produto();
84                  newProduto.setIdProduto(idProduto);
85                  newProduto.setNome(nome);
86                  newProduto.setQuantidade(quantidade);
87                  newProduto.setPrecoVenda(precoVenda);
88
89                  facade.create(newProduto);
90
91                  List<Produto> newProdutos = facade.findAll();
92                  request.setAttribute("produtos", newProdutos);
93                  break;
94
95              case "alterar":
96                  Produto alterarProduto = facade.find(Integer.valueOf(request.getParameter("id")));
97
98                  String alterarNome = request.getParameter("nome");
99                  int alterarQuantidade = Integer.parseInt(request.getParameter("quantidade"));
100                  Float alterarPrecoVenda = Float.valueOf(request.getParameter("precoVenda"));
101
102                  alterarProduto.setNome(alterarNome);
103                  alterarProduto.setQuantidade(alterarQuantidade);
104                  alterarProduto.setPrecoVenda(alterarPrecoVenda);
105
106                  facade.edit(alterarProduto);
107                  List<Produto> alterarProdutos = facade.findAll();
108                  request.setAttribute("produtos", alterarProdutos);
109                  break;
110
111              default:
112                  List<Produto> produtos = facade.findAll();
113                  request.setAttribute("produtos", produtos);
114                  break;
115          }
116
117      RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
118      dispatcher.forward(request, response);
119  }
```

- ✓ Criar a página de consulta, com o nome **ProdutoLista.jsp**
- ✓ Incluir um link para **ServletProdutoFC**, com acao **formIncluir**, voltado para a abertura do formulário de inclusão.
- ✓ Definir uma tabela para apresentação dos dados.
- ✓ Recuperar a **lista de produtos** enviada pelo Servlet.
- ✓ Para cada elemento da lista, apresentar id, nome, quantidade e preço como células da tabela.
- ✓ Criar, também, de forma dinâmica, links para **alteração** e **exclusão**,
- ✓ com a chamada para ServletProdutoFC, passando as ações corretas e o id do elemento corrente.
- ✓ Organizar o código para obter uma página como a seguinte.



ID	Produto	Quantidade	Preço	Ações
1	Cubo de Roda	100	R\$ 60.00	Alterar Excluir
3	Disco de Freio	200	R\$ 290.00	Alterar Excluir
4	Pastilha de Freio	400	R\$ 55.00	Alterar Excluir

[Cadastrar Produto](#)

- ✓ Criar a página de cadastro, com o nome **ProdutoDados.jsp**
- ✓ Definir um formulário com envio para **ServletProdutoFC**, modo **post**.
- ✓ Recuperar a **entidade** enviada pelo Servlet.
- ✓ Definir a variável **acao**, com valor **incluir**, para entidade nula, ou **alterar**, quando a entidade é fornecida.
- ✓ Incluir um campo do tipo **hidden**, para envio do valor de **acao**.
- ✓ Incluir um campo do tipo **hidden**, para envio do **id**, apenas quando o valor de **acao** for **alterar**.
- ✓ Incluir os campos para nome, quantidade e preço de venda, preenchendo os dados quando a entidade é fornecida.
- ✓ Concluir o formulário com um botão de envio, com o texto adequado para as situações de inclusão ou alteração de dados.
- ✓ Organizar o código para obter uma página como a seguinte.

Cadastro de Produto

[Voltar](#)

ID:

Nome:

Quantidade:

Preço de Venda:

Os resultados da execução dos códigos também devem ser Apresentados:

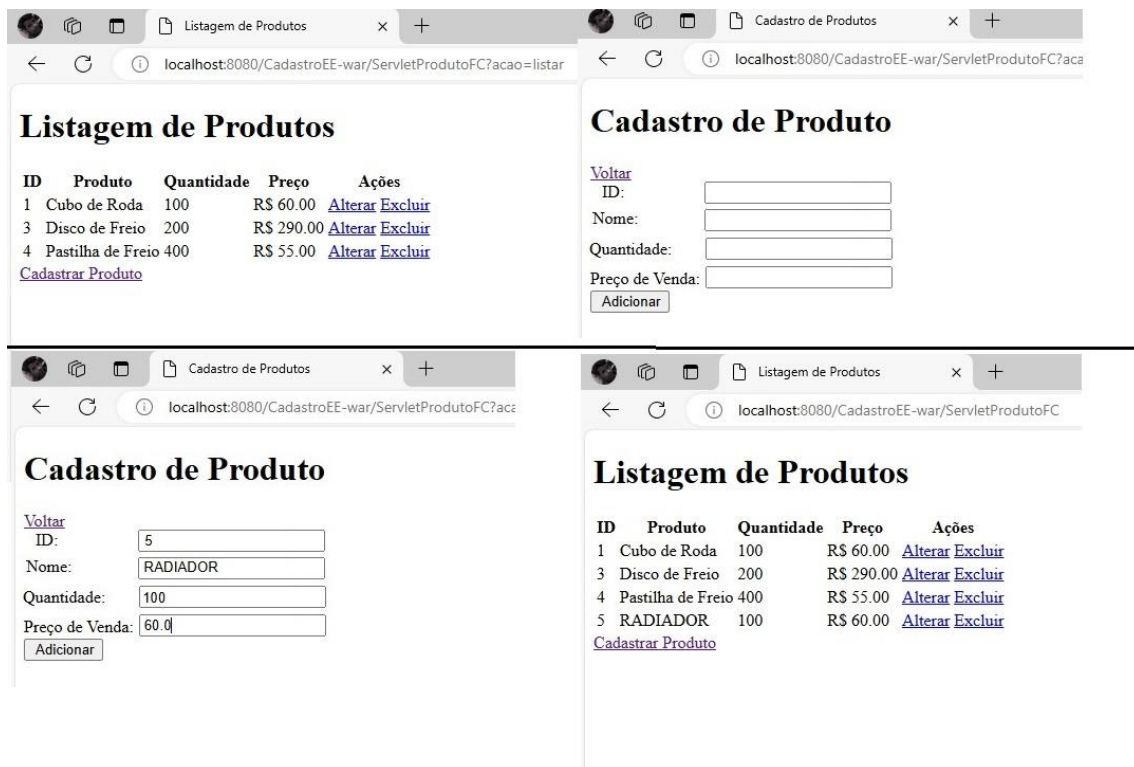
Codigo do ProdutoListas.jsp

```
1 <%@ page import="java.text.DecimalFormat" %>
2 <%@ page import="cadastroes.model.Produto" %>
3 <%@ page import="java.util.List" %>
4 <%@ page contentType="text/html" pageEncoding="UTF-8" %>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <title>Listagem de Produtos</title>
9
10
11 </head>
12 <body>
13 <div class="container">
14 <div class="header-section text-center">
15 <h1>Listagem de Produtos</h1>
16 </div>
17
18 <div class="card">
19 <div class="card-body">
20
21
22 <table class="table table-striped table-bordered table-responsive">
23 <thead>
24 <tr class="table-dark">
25 <th>ID</th>
26 <th>Produto</th>
27 <th>Quantidade</th>
28 <th>Preço</th>
29 <th>Ações</th>
30 </tr>
31 </thead>
32
33 <%
34 DecimalFormat df = new DecimalFormat("#,##0.00");
35 List<Produto> produtos = (List<Produto>) request.getAttribute("produtos");
36
37 if (produtos != null && !produtos.isEmpty()) {
38     for (Produto produto : produtos) {
39
40 <tr>
41 <td class="text-center"><%=produto.getIdProduto() %></td>
42 <td class="text-center"><%=produto.getNome() %></td>
43 <td class="text-center"><%=produto.getQuantidade() %></td>
44 <td class="text-center">R$ <%=df.format(produto.getPrecoVenda()) %></td>
45 <td class="text-end">
46 <a class="btn btn-primary btn-sm" href="ServletProdutoFC?acao=formAlterar&id=<%=produto.getIdProduto() %>">Alterar</a>
47 <a class="btn btn-danger btn-sm" href="ServletProdutoFC?acao=excluir&id=<%=produto.getIdProduto() %>">Excluir</a>
48 </td>
49 </tr>
50 <%
51     }
52 } else {
53 <tr>
54 <td colspan="5">Nenhum produto encontrado.</td>
55 </tr>
56 <%
57     }
58 }
59 </table>
60
61 <div class="text-end mb-3">
62 <a class="btn btn-primary" href="ServletProdutoFC?acao=formIncluir">Cadastrar Produto</a>
63 </div>
64 </div>
65 </div>
66 </body>
67 </html>
```

Codigo do ProdutoDados.jsp

```
1 <%@page import="cadastroee.model.Produto"%>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7 <title>Cadastro de Produtos</title>
8
9 </head>
10 <body class="container">
11
12 <%
13     Produto produto = (Produto) request.getAttribute("produto");
14     String acao = produto != null ? "alterar" : "incluir";
15 %>
16
17 <h1><%=acao == "alterar" ? "Alteração" : "Cadastro"%> de Produto</h1>
18 <a class="btn btn-secondary" href="ServletProdutoFC?acao=listar">Voltar</a>
19 <br><br>
20 <div>
21     <form class="form" action="ServletProdutoFC" method="post">
22         <input type="hidden" name="acao" value="<%=acao%>">
23         <% if (produto != null) {%>
24         <input type="hidden" name="id" value="<%=produto.getIdProduto()%>">
25         <% }%>
26         <div class="mb-3">
27             <label class="form-label" for="idProduto">ID:</label>
28             <input class="form-control" type="text" id="idProduto" name="idProduto" required>
29         </div>
30
31         <div class="mb-3">
32             <label class="form-label" for="nome">Nome:</label>
33             <input class="form-control" type="text" name="nome" value="<%=produto != null ? produto.getNome() : ""%>" required>
34         </div>
35
36         <div class="mb-3">
37             <label class="form-label" for="quantidade">Quantidade:</label>
38             <input class="form-control" type="text" name="quantidade" value="<%=produto != null ? produto.getQuantidade() : ""%>" required>
39         </div>
40
41         <div class="mb-3">
42             <label class="form-label" for="precoVenda">Preço de Venda:</label>
43             <input class="form-control" type="text" name="precoVenda" value="<%=produto != null ? produto.getPrecoVenda() : ""%>" required>
44         </div>
45
46         <div>
47             <input class="btn btn-primary" type="submit" value="<%=acao == "incluir" ? "Adicionar" : "Alterar"%>">
48         </div>
49     </form>
50 </div>
51 </body>
52 </html>
53
```

Resultado:



5 - Análise e Conclusão:

❖ Como funciona o padrão Front Controller, e como ele é implementado em um aplicativo Web Java, na arquitetura MVC?

- ✓ O padrão Front Controller funciona como um ponto central de entrada para todas as requisições em uma aplicação web, direcionando-as para os controladores específicos que processam a lógica necessária. Em uma arquitetura MVC com Java, é implementado geralmente através de um Servlet que captura todas as requisições, decide qual controlador deve processá-las e, em seguida, despacha a requisição para o controlador correspondente. Isso centraliza o controle, simplifica o tratamento de requisições e facilita a manutenção. Segue abaixo Os Benefícios do Padrão Front Controller.

Benefícios do Padrão Front Controller

- ✓ **Centralização:** Todas as requisições passam por um ponto central, facilitando a gestão de segurança, autenticação, autorização e outros filtros.
- ✓ **Facilidade de Manutenção:** Modificações no fluxo de requisições podem ser feitas de forma centralizada, sem a necessidade de alterar cada controlador específico.
- ✓ **Consistência:** Garantia de que todas as requisições sejam tratadas de maneira consistente.
- ✓ Ao implementar o padrão Front Controller em uma aplicação Java Web com MVC, você obtém uma estrutura organizada e eficiente para gerenciar requisições, resultando em uma aplicação mais robusta e fácil de manter.

❖ Quais as diferenças e semelhanças entre Servlets e JSPs?

- ✓ **Servlets e JSPs** são tecnologias Java usadas no desenvolvimento de aplicações web, e enquanto compartilham algumas semelhanças, têm propósitos e funcionalidades distintas.

❖ Semelhanças

- ✓ **Baseadas em Java:** Ambas são componentes Java que rodam no servidor e são parte da especificação Java EE para desenvolvimento de aplicações web.
- ✓ **Geram HTML Dinâmico:** Tanto Servlets quanto JSPs são usadas para gerar conteúdo HTML dinâmico, que é enviado ao cliente (navegador).
- ✓ **Integração com MVC:** Ambos podem ser utilizados em conjunto para implementar o padrão MVC (Model-View-Controller), onde os Servlets frequentemente atuam como controladores e as JSPs como views.

❖ Diferenças

- ✓ **Propósito Principal:**
 - **Servlets:** São mais orientados à lógica de controle e processamento de requisições. Eles são classes Java que manipulam requisições HTTP, executam lógica de negócio e direcionam a resposta para o cliente.
 - **JSPs (JavaServer Pages):** São mais orientadas à apresentação, ou seja, para a geração de páginas HTML. São páginas que misturam HTML com código Java embutido para dinamizar o conteúdo.

❖ Sintaxe e Facilidade de Uso:

- **Servlets:** O código Java é predominantemente usado, o que pode tornar a escrita de código para a geração de HTML mais complicada e menos intuitiva.
- **JSPs:** Usam uma sintaxe que mistura HTML com tags JSP e EL (Expression Language), facilitando a criação de páginas dinâmicas. O código Java pode ser incluído diretamente na página, embora a prática moderna recomende minimizar isso.

❖ Ciclo de Vida:

- **Servlets:** São compilados diretamente em bytecode Java e executados pelo servidor de aplicações.
- **JSPs:** São inicialmente compiladas em Servlets pelo servidor de aplicações, mas essa etapa é transparente para o desenvolvedor. O resultado final é executado como um Servlet.

❖ Responsabilidade no MVC:

- **Servlets:** Geralmente atuam como o "Controlador", recebendo as requisições, processando dados e decidindo qual página JSP deve ser exibida.
 - **JSPs:** Atuam como a "View", responsável por renderizar a interface do usuário.
- ✓ Essencialmente, enquanto **Servlets** são ideais para manipulação lógica e controle, **JSPs** são mais adequadas para a geração de conteúdo visual.


❖ Qual a diferença entre um redirecionamento simples e o uso do método forward, a partir do RequestDispatcher? Para que servem parâmetros e atributos nos objetos HttpRequest?

Diferença entre Redirecionamento Simples e RequestDispatcher.forward()

❖ Redirecionamento Simples (HTTP Redirect):

- **Como funciona:** O servidor envia uma resposta ao cliente (navegador) com um código de status HTTP 3xx (geralmente 302) e uma nova URL para onde o navegador deve redirecionar. O navegador, então, faz uma nova requisição para essa URL.
- **Efeito:** O cliente vê a nova URL no navegador, e uma nova requisição HTTP é feita, resultando em dois ciclos de requisição-resposta.
- **Uso:** Usado quando se quer redirecionar o cliente para uma URL diferente, seja dentro da mesma aplicação ou para uma aplicação externa.

java


 Copiar código

```
response.sendRedirect("novaPagina.jsp");
```

❖ RequestDispatcher.forward():

- **Como funciona:** O encaminhamento (forward) é realizado internamente no servidor. O controle da requisição é passado de um servlet (ou JSP) para outro recurso (como outro servlet ou JSP) sem que o cliente saiba disso.
- **Efeito:** O cliente não vê a mudança na URL, e apenas uma única requisição HTTP é processada no servidor.
- **Uso:** Usado para passar a requisição para outro recurso no servidor, geralmente dentro da mesma aplicação, mantendo o estado da requisição.

java

 Copiar código

```
RequestDispatcher dispatcher = request.getRequestDispatcher("novaPagina.jsp");  
dispatcher.forward(request, response);
```

❖ Parâmetros e Atributos no Objeto HttpServletRequest

❖ Parâmetros (request.getParameter ()):

- **O que são:** Dados enviados pelo cliente ao servidor, geralmente através de formulários HTML via método GET ou POST. Cada parâmetro é um par nome-valor.
- **Uso:** Utilizados para capturar dados enviados na URL ou no corpo da requisição (como informações de um formulário).
- **Exemplo:**

```
java Copiar código  
  
String nome = request.getParameter("nome");
```

❖ Atributos (request.setAttribute () e request.getAttribute ()):

- **O que são:** Objetos que podem ser armazenados na requisição para serem compartilhados entre diferentes recursos (servlets ou JSPs) durante o ciclo de vida da requisição. Atributos não são enviados pelo cliente, mas sim adicionados no servidor.
- **Uso:** Utilizados para passar dados entre servlets e JSPs durante o processamento da mesma requisição.
- **Exemplo:**

```
java Copiar código  
  
request.setAttribute("usuario", usuario);  
Usuario usuario = (Usuario) request.getAttribute("usuario");
```

❖ Resumo:

- O redirecionamento simples resulta em uma nova requisição HTTP e altera a URL visível, enquanto o forward mantém a mesma requisição e não altera a URL no navegador. Parâmetros são usados para dados enviados pelo cliente, e atributos são usados para compartilhar dados no servidor durante o processamento da requisição.

👉 3º Procedimento | Melhorando o Design da Interface

1- Objetivo da Prática:

- Implementar persistência com base em JPA.
- Implementar regras de negócio na plataforma JEE, através de EJBs.
- Implementar sistema cadastral Web com base em Servlets e JSPs.
- Utilizar a biblioteca Bootstrap para melhoria do design.
- No final do exercício, o aluno terá criado todos os elementos necessários para exibição e entrada de dados na plataforma Java Web, tornando-se capacitado para lidar com contextos reais de aplicação.



- ✓ Incluir as bibliotecas do framework Bootstrap nos arquivos ProdutoLista.jsp e ProdutoDados.jsp
- ✓ Visitar o site do Bootstrap, no endereço <https://getbootstrap.com/>
- ✓ Rolar para baixo até encontrar a inclusão via CDN
- ✓ Clicar no botão para cópia do link **CSS** e colar na divisão **head** de cada uma das páginas JSP.
- ✓ Clicar no botão para cópia do link para a biblioteca **Java Script** e colar na divisão **head** de cada uma das páginas JSP



ID	Produto	Quantidade	Preço	Ações
1	Cubo de Roda	100	R\$ 60.00	Alterar Excluir
3	Disco de Freio	200	R\$ 290.00	Alterar Excluir
4	Pastilha de Freio	400	R\$ 55.00	Alterar Excluir

[Cadastrar Produto](#)

- ✓ Modificar as características de **ProdutoLista.jsp**
- ✓ Adicionar a classe **container** ao body.
- ✓ Adicionar as classes **btn**, **btn-primary** e **m-2** no link de inclusão.
- ✓ Adicionar as classes **table** e **table-striped** na tabela.
- ✓ Adicionar a classe **table-dark** ao thead.
- ✓ Adicionar as classes **btn**, **btn-primary** e **btn-sm** no link de alteração.
- ✓ Adicionar as classes **btn**, **btn-danger** e **btn-sm** no link de exclusão.
- ✓ Ajustar as características para obter o design apresentado a seguir.



ID	Produto	Quantidade	Preço	Ações
1	Cubo de Roda	100	R\$ 60,00	Alterar Excluir
3	Disco de Freio	200	R\$ 290,00	Alterar Excluir
4	Pastilha de Freio	400	R\$ 55,00	Alterar Excluir

[Cadastrar Produto](#)

- ✓ Modificar as características de **ProdutoDados.jsp**
- ✓ Adicionar a classe **container** ao body.
- ✓ Encapsule cada par label / input em **div** com classe **mb-3**.
- ✓ Adicionar a classe **form** ao formulário.
- ✓ Adicionar a classe **form-label** em cada label.
- ✓ Adicionar a classe **form-control** em cada input.
- ✓ Adicionar as classes **btn** e **btn-primary** ao botão de inclusão.
- ✓ Ajustar as características para obter o design apresentado a seguir.

Alteração de Produtos

Voltar

ID: 6

Nome: Radiador

Quantidade: 300

Preço de Venda: 255.0

Alterar

Listagem de Produtos

ID	Produto	Quantidade	Preço	Ações
1	Cubo de Roda	100	R\$ 60,00	Alterar Excluir
3	Disco de Freio	200	R\$ 290,00	Alterar Excluir
4	Pastilha de Freio	400	R\$ 55,00	Alterar Excluir
5	Radiador	200	R\$ 155,00	Alterar Excluir

Cadastrar Produto

Todos os códigos solicitados neste roteiro de aula:

Código do ProdutoLista.jsp com as Devidas alterações solicitadas.

```
1 <%@ page import="java.text.DecimalFormat" %>
2 <%@ page import="Cadastro.modelo.Produto" %>
3 <%@ page import="java.util.List" %>
4 <%@ page contentType="text/html" pageEncoding="UTF-8" %>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <title>Listagem de Produtos</title>
9 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
10 <style>
11 body {
12     background-color: #f4f4f4;
13     font-family: 'Arial', sans-serif;
14 }
15 .header-section {
16     background-color: #007bff;
17     color: white;
18     padding: 20px 0;
19     margin-bottom: 30px;
20 }
21 .header-section h1 {
22     margin: 0;
23     font-size: 2.5rem;
24 }
25 .card {
26     background-color: white;
27     padding: 20px;
28     border-radius: 8px;
29     box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
30 }
31 .table {
32     background-color: #007bff;
33     color: white;
34 }
35 .table-striped tbody tr:nth-of-type(odd) {
36     background-color: rgba(0, 123, 255, .1);
37 }
38 .btn-primary {
39     background-color: #007bff;
40     border-color: #007bff;
41 }
42 .btn-danger {
43     background-color: #dc3545;
44     border-color: #dc3545;
45 }
46 .btn-sm {
47     padding: .25rem .5rem;
48     font-size: .875rem;
49     line-height: 1.5;
50     border-radius: .2rem;
51 }
52 </style>
53 </head>
54 <body>
55 <div class="container">
56 <div class="header-section text-center">
57 <h1>Listagem de Produtos</h1>
58 </div>
59 <div class="card">
60 <div class="card-body">
61
62
63
64 <table class="table table-striped table-bordered table-responsive">
65 <thead>
66 <tr class="table-dark">
67 <th>ID</th>
68 <th>Produto</th>
69 <th>Quantidade</th>
70 <th>Preço</th>
71 <th>Ações</th>
72 </tr>
73 </thead>
74
75 <%
76     DecimalFormat df = new DecimalFormat("#,##0.00");
77     List<Produto> produtos = (List<Produto>) request.getAttribute("produtos");
78
79     if (produtos != null && !produtos.isEmpty()) {
80         for (Produto produto : produtos) {
81
82 <tr>
83 <td class="text-center"><%=produto.getIdProduto()%></td>
84 <td class="text-center"><%=produto.getNome()%></td>
85 <td class="text-center"><%=produto.getQuantidade()%></td>
86 <td class="text-center">R$ <%=df.format(produto.getPrecoVenda())%></td>
87 <td class="text-center">
88 <a class="btn btn-primary btn-sm" href="ServletProdutoFC?acao=formAlterar&id=<%=produto.getIdProduto()%>">Alterar</a>
89 <a class="btn btn-danger btn-sm" href="ServletProdutoFC?acao=excluir&id=<%=produto.getIdProduto()%>">Excluir</a>
90 </td>
91 </tr>
92 <%
93         }
94     } else {
95 <tr>
96 <td colspan="5">Nenhum produto encontrado.</td>
97 </tr>
98 <%
99     }
100 <%>
101 </table>
102
103 <div class="text-end mb-3">
104 <a class="btn btn-primary" href="ServletProdutoFC?acao=formIncluir">Cadastrar Produto</a>
105 </div>
106 </div>
107 </div>
108 </div>
109 </body>
110 </html>
```

Código do ProdutoDados.jsp com as Devidas alterações solicitadas.

```
1  <%@page import="cadastroee.model.Produto"%>
2  <%@page contentType="text/html" pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7      <title>Cadastro de Produtos</title>
8      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9      <style>
10         body {
11             background-color: #f4f4f4;
12             font-family: 'Arial', sans-serif;
13         }
14         .header-section {
15             background-color: #007bff;
16             color: white;
17             padding: 20px 0;
18             margin-bottom: 30px;
19         }
20         .header-section h1 {
21             margin: 0;
22             font-size: 2.5rem;
23         }
24         .form-container {
25             background-color: white;
26             padding: 20px;
27             border-radius: 8px;
28             box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
29         }
30         .form-label {
31             font-weight: bold;
32             color: #333;
33         }
34         .btn-primary {
35             background-color: #007bff;
36             border-color: #007bff;
37         }
38         .btn-secondary {
39             margin-bottom: 15px;
40         }
41     </style>
42 </head>
43 <body class="container">
44     <%
45         Produto produto = (Produto) request.getAttribute("produto");
46         String acao = produto != null ? "alterar" : "incluir";
47     %>
48
49     <h1><%=acao == "alterar" ? "Alteração" : "Cadastro"%> de Produto</h1>
50     <a class="btn btn-secondary" href="ServletProdutoFC/acao=listar">Voltar</a>
51     <br><br>
52     <div>
53         <form class="form" action="ServletProdutoFC" method="post">
54             <input type="hidden" name="acao" value="<%=acao%>">
55             <% if (produto != null) {%>
56                 <input type="hidden" name="id" value="<%=produto.getIdProduto()%>">
57                 <% }%>
58             <div class="mb-3">
59                 <label class="form-label" for="idProduto">ID:</label>
60                 <input class="form-control" type="text" id="idProduto" name="idProduto" required>
61             </div>
62             <div class="mb-3">
63                 <label class="form-label" for="nome">Nome:</label>
64                 <input class="form-control" type="text" name="nome" value="<%=produto != null ? produto.getNome() : ""%>" required>
65             </div>
66             <div class="mb-3">
67                 <label class="form-label" for="quantidade">Quantidade:</label>
68                 <input class="form-control" type="text" name="quantidade" value="<%=produto != null ? produto.getQuantidade() : ""%>" required>
69             </div>
70             <div class="mb-3">
71                 <label class="form-label" for="precoVenda">Preço de Venda:</label>
72                 <input class="form-control" type="text" name="precoVenda" value="<%=produto != null ? produto.getPrecoVenda() : ""%>" required>
73             </div>
74             <div>
75                 <input class="btn btn-primary" type="submit" value="<%=acao == "incluir" ? "Adicionar" : "Alterar"%>">
76             </div>
77         </form>
78     </div>
79     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
80 </body>
81 </html>
```


Os resultados da execução dos códigos também devem ser apresentados:

Pagina com a Listagem dos produtos como está no Banco SQL

localhost:8080/CadastroEE-war/ServletProdutoFC

Listagem de Produtos

ID	Produto	Quantidade	Preço	Ações
1	Cubo de Roda	100	R\$ 60,00	<button>Alterar</button> <button>Excluir</button>
3	Disco de Freio	200	R\$ 290,00	<button>Alterar</button> <button>Excluir</button>
4	Pastilha de Freio	400	R\$ 55,00	<button>Alterar</button> <button>Excluir</button>
5	Radiador	200	R\$ 155,00	<button>Alterar</button> <button>Excluir</button>

Cadastrar Produto

SQLQuery1.sql - USU-PQNOLLI7GBQ\TIAGOPF.Loja (loja (65)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

Loja Execute

SQL Shades

Object Explorer

Connect

USU-PQNOLLI7GBQ\TIAGOPF

Databases

System Databases

Database Snapshots

Loja

Database Diagrams

Tables

System Tables

FileTables

External Tables

Graph Tables

dbo.Movimento

dbo.Pessoa

dbo.PessoaFisica

dbo.PessoaJuridica

dbo.Produto

dbo.Usuario

Views

External Resources

Synonyms

Programmability

Service Broker

Storage

Security

SQLQuery1.sql - USU...PF.Loja (loja (65))

SELECT TOP (1000) [idProduto]

[nome]

[quantidade]

[precoVenda]

FROM [Loja].[dbo].[Produto]

107 %

Results Messages

	idProduto	nome	quantidade	precoVenda
1	1	Cubo de Roda	100	60.00
2	3	Disco de Freio	200	290.00
3	4	Pastilha de Freio	400	55.00
4	5	Radiador	200	155.00

Query executed successfully.

Pagina com de Alteração de Produtos com exemplo e logo abaixo o resultado no servidor SQL

localhost:8080/CadastroEE-war/ServletProdutoFC?acao=formAlterar&id=5

Alteração de Produtos

Voltar

ID:
6

Nome:
Radiador

Quantidade:
300

Preço de Venda:
255.0

Alterar

SQLQuery1.sql - USU-PQNOLLI7GBQ\TIAGOPF.Loja (loja (65)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query MDX DMX XMLA DAX

Loja Execute

SQL Shades

Object Explorer

Connect

USU-PQNOLLI7GBQ\TIAGOPF

Databases

System Databases

Database Snapshots

Loja

Database Diagrams

Tables

System Tables

FileTables

External Tables

Graph Tables

dbo.Movimento

dbo.Pessoa

dbo.PessoaFisica

dbo.PessoaJuridica

dbo.Produto

dbo.Usuario

Views

External Resources

Synonyms

Programmability

Service Broker

Storage

Security

SQLQuery1.sql - USU...PF.Loja (loja (65))

SELECT TOP (1000) [idProduto]
, [nome]
, [quantidade]
, [precoVenda]
FROM [Loja].[dbo].[Produto]

107 %

Results Messages

	idProduto	nome	quantidade	precoVenda
1	1	Cubo de Roda	100	60.00
2	3	Disco de Freio	200	290.00
3	4	Pastilha de Freio	400	55.00
4	6	Radiador	300	255.00

Query executed successfully.

Pagina de Cadastro de Produto com exemplo e logo abaixo o resultado no servidor SQL

Cadastro de Produtos

Voltar

ID:

7

Nome:

Fluido Paraflu

Quantidade:

1500

Preço de Venda:

45.00

Adicionar

SQLQuery1.sql - USU-PQNOLLI7GBQ\TIAGOPF.Loja (loja (65)) - Microsoft SQL Server Management Studio

FileEditViewQueryProjectToolsWindowHelp

LojaExecute

SQL Shades

Object Explorer

Connect

USU-PQNOLLI7GBQ\TIAGOPF

Databases

System Databases

Database Snapshots

Loja

Database Diagrams

Tables

System Tables

FileTables

External Tables

Graph Tables

dbo.Movimento

dbo.Pessoa

dbo.PessoaFisica

dbo.PessoaJuridica

dbo.Produto

dbo.Usuario

Views

External Resources

Synonyms

Programmability

Service Broker

Storage

Security

SQLQuery1.sql - USU...PF.Loja (loja (65))

SELECT TOP (1000) [idProduto]
,[nome]
,[quantidade]
,[precoVenda]
FROM [Loja].[dbo].[Produto]

107 %

ResultsMessages

	idProduto	nome	quantidade	precoVenda
1	1	Cubo de Roda	100	60.00
2	3	Disco de Freio	200	290.00
3	4	Pastilha de Freio	400	55.00
4	6	Radiador	300	255.00
5	7	Fluido Paraflu	1500	45.00

Query executed successfully.

5- Análise e Conclusão:

❖ Como o framework Bootstrap é utilizado?

- ✓ O Bootstrap é um framework front-end bastante popular que facilita a criação de sites e aplicações web responsivas e modernas. Ele oferece uma coleção de ferramentas de design, como layout grid, componentes prontos, e uma vasta biblioteca de classes CSS e scripts JavaScript que permitem que os desenvolvedores criem interfaces de utilizador de forma mais rápida e eficiente.

❖ Aqui está um resumo de como o Bootstrap é utilizado:

❖ Estrutura de Layout:

- ✓ O Bootstrap utiliza um sistema de grid baseado em 12 colunas, permitindo que os desenvolvedores criem layouts responsivos que se adaptam a diferentes tamanhos de ecrã. Os elementos são organizados em linhas (rows) e colunas (cols), que podem ser ajustadas conforme necessário.

❖ Componentes Prontos:

- ✓ O Bootstrap vem com uma vasta gama de componentes UI prontos, como botões, formulários, barras de navegação, carrosséis, modais, alertas, entre outros. Estes componentes são altamente personalizáveis através de classes CSS.

❖ Responsividade:

- ✓ O Bootstrap é projetado para ser responsivo por padrão. Isso significa que os sites construídos com Bootstrap se ajustam automaticamente a diferentes tamanhos de ecrã (desde dispositivos móveis até computadores de secretária) sem necessidade de escrever CSS adicional.

❖ Personalização:

- ✓ Embora o Bootstrap ofereça estilos predefinidos, ele também permite a personalização através de variáveis Sass e classes utilitárias. Os desenvolvedores podem modificar facilmente cores, tipografia, espaçamento e outros aspectos de design para se adequarem às necessidades específicas do projeto.

❖ Integração com JavaScript:

- ✓ O Bootstrap inclui vários plugins JavaScript que adicionam funcionalidades interativas, como dropdowns, modais, tooltips, entre outros. Estes plugins são fáceis de usar e podem ser integrados sem precisar de muito código adicional.

❖ Documentação:

- ✓ O Bootstrap é conhecido pela sua excelente documentação, que facilita a aprendizagem e implementação das suas funcionalidades. A documentação oficial fornece exemplos detalhados e explicações de como usar cada componente e funcionalidade.

❖ Exemplo básico de utilização:

- ✓ Aqui está um exemplo simples de como usar o Bootstrap para criar uma página com um layout básico:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo Bootstrap</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-4">
        <h2>Coluna 1</h2>
        <p>Conteúdo da primeira coluna.</p>
      </div>
      <div class="col-md-4">
        <h2>Coluna 2</h2>
        <p>Conteúdo da segunda coluna.</p>
      </div>
      <div class="col-md-4">
        <h2>Coluna 3</h2>
        <p>Conteúdo da terceira coluna.</p>
      </div>
    </div>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

- ✓ Neste exemplo, a página usa o grid do Bootstrap para dividir o conteúdo em três colunas, que são responsivas e se ajustam conforme o tamanho do ecrã.
- ✓ O Bootstrap é ideal para desenvolvedores que desejam criar interfaces de utilizador elegantes e responsivas de maneira eficiente, sem precisar construir tudo do zero.

❖ Por que o Bootstrap garante a independência estrutural do HTML?

- ✓ O Bootstrap garante a independência estrutural do HTML ao fornecer uma estrutura e classes CSS que separam o design visual do conteúdo. Isso significa que o conteúdo HTML pode ser gerido e modificado sem precisar alterar a estrutura visual ou a aparência do site. Existem várias razões pelas quais o Bootstrap garante essa independência:

❖ Classes CSS Padronizadas:

- ✓ O Bootstrap oferece uma grande variedade de classes CSS pré-definidas que podem ser aplicadas diretamente aos elementos HTML. Isso permite que os desenvolvedores alterem a aparência e o comportamento dos elementos sem modificar o HTML subjacente. Por exemplo, para transformar um botão normal num botão com aparência profissional, basta adicionar a classe `btn btn-primary` ao elemento `button`.

❖ Sistema de Grid:

- ✓ O sistema de grid do Bootstrap permite que os desenvolvedores organizem o layout da página usando classes CSS (`container`, `row`, `col`). Este sistema de grid é totalmente separado do conteúdo, o que significa que a disposição dos elementos pode ser alterada ajustando apenas as classes CSS, sem a necessidade de reescrever o HTML.

❖ Responsividade Integrada:

- ✓ O Bootstrap incorpora princípios de design responsivo diretamente nas suas classes e componentes. As classes como `col-md-4` ou `col-sm-6` fazem com que os elementos se ajustem automaticamente a diferentes tamanhos de ecrã, garantindo que o layout se adapta sem modificar o conteúdo HTML.

❖ Componentes Reutilizáveis:

- ✓ O Bootstrap fornece componentes de interface reutilizáveis (como modais, dropdowns e carrosséis) que podem ser implementados em várias partes do site sem repetir ou alterar a estrutura HTML. Esses componentes são altamente configuráveis através de classes CSS e atributos de dados, garantindo que o design seja independente do conteúdo.

❖ **Consistência de Design:**

- ✓ Com o Bootstrap, a consistência visual do site é garantida através do uso de um conjunto comum de estilos. Isso significa que os elementos HTML podem ser estruturados de maneiras diferentes, mas terão uma aparência consistente em toda a aplicação graças às classes padronizadas do Bootstrap.

❖ **Facilidade de Manutenção e Escalabilidade:**

- ✓ Separar a estrutura visual (CSS) do conteúdo (HTML) facilita a manutenção do código. Alterações no design ou na aparência do site podem ser feitas atualizando as classes CSS sem tocar na estrutura HTML. Isso torna o desenvolvimento mais eficiente e escalável.

❖ **Exemplo de Independência Estrutural:**

- ✓ Considere dois botões em HTML:

```
html Copiar código  
  
<button class="btn btn-primary">Botão 1</button>  
<button class="btn btn-secondary">Botão 2</button>
```

- ✓ Os botões têm a mesma estrutura HTML (<button>), mas usam diferentes classes Bootstrap para ter estilos visuais distintos (btn-primary e btn-secondary). Se precisar alterar o estilo, você só precisa ajustar as classes CSS, mantendo o HTML inalterado.
- ✓ **Conclusão:**
- ✓ O Bootstrap garante a independência estrutural do HTML ao fornecer uma arquitetura onde a apresentação visual (CSS) é separada do conteúdo e da estrutura HTML. Isso permite maior flexibilidade, manutenibilidade e eficiência no desenvolvimento de aplicações web.

❖ Qual a relação entre o Bootstrap e a responsividade da página?

- ✓ A relação entre o Bootstrap e a responsividade da página é fundamental, pois um dos principais objetivos do Bootstrap é facilitar a criação de sites e aplicações web que se adaptem automaticamente a diferentes tamanhos de ecrã e dispositivos, proporcionando uma experiência de utilizador consistente e agradável, independentemente do dispositivo utilizado.
- ✓ Aqui estão alguns pontos-chave que explicam como o Bootstrap lida com a responsividade:

❖ Sistema de Grid Responsivo:

- ✓ O Bootstrap utiliza um sistema de grid flexível e baseado em colunas que se adapta automaticamente ao tamanho do ecrã. Este sistema divide a página em 12 colunas, permitindo que os desenvolvedores criem layouts que se reorganizam e redimensionam conforme o dispositivo.
- ✓ As classes como col-sm-, col-md-, col-lg-, e col-xl- especificam como os elementos devem se comportar em diferentes tamanhos de ecrã (pequeno, médio, grande e extra grande). Por exemplo, col-md-6 ocupa metade do ecrã em dispositivos médios e maiores, mas pode ser ajustado para ocupar todo o ecrã em dispositivos pequenos (col-sm-12).

❖ Breakpoints Responsivos:

- ✓ O Bootstrap define breakpoints específicos que correspondem a diferentes larguras de ecrã, como 576px (para dispositivos pequenos), 768px (para dispositivos médios), 992px (para dispositivos grandes), e 1200px (para dispositivos extra grandes). Esses breakpoints permitem que o layout mude automaticamente para se ajustar ao tamanho do ecrã, utilizando as classes apropriadas.

❖ Classes Utilitárias:

- ✓ O Bootstrap inclui uma vasta gama de classes utilitárias que podem ser usadas para aplicar estilos responsivos diretamente nos elementos HTML. Por exemplo, as classes como d-none e d-md-block podem ser usadas para esconder um elemento em ecrãs pequenos e mostrá-lo apenas em ecrãs médios e maiores.

❖ Componentes Responsivos:

- ✓ Muitos dos componentes do Bootstrap, como navbars, carrosséis e modais, são projetados para serem responsivos. Eles ajustam automaticamente seu comportamento e aparência de acordo com o tamanho do ecrã. Por exemplo, a navbar pode se transformar num menu tipo "hambúrguer" em dispositivos móveis, garantindo uma navegação fácil em ecrãs menores.

❖ Imagens e Media Responsivas:

- ✓ O Bootstrap facilita a implementação de imagens e outros media que se adaptam ao tamanho do ecrã. A classe `img-fluid`, por exemplo, faz com que as imagens se redimensionem automaticamente para caber na largura do container, evitando que elas fiquem maiores do que o ecrã do utilizador.

❖ Layouts Mobile-First:

- ✓ O Bootstrap segue uma abordagem "mobile-first", o que significa que o design é inicialmente concebido para dispositivos móveis e, em seguida, são aplicadas as media queries e outros ajustes para melhorar a experiência em dispositivos maiores. Isso garante que o site funcione bem em dispositivos móveis, que representam uma parte significativa dos acessos à web hoje em dia.

❖ Exemplo de Layout Responsivo:

- ✓ Aqui está um exemplo de como o sistema de grid do Bootstrap pode ser usado para criar um layout responsivo:

```
1 <div class="container">
2   <div class="row">
3     <div class="col-sm-12 col-md-8">
4       <h2>Conteúdo Principal</h2>
5       <p>Este conteúdo ocupa 100% da largura em dispositivos pequenos e 8 colunas em dispositivos médios e maiores.</p>
6     </div>
7     <div class="col-sm-12 col-md-4">
8       <h2>Barra Lateral</h2>
9       <p>Esta barra lateral ocupa 100% da largura em dispositivos pequenos e 4 colunas em dispositivos médios e maiores.</p>
10    </div>
11  </div>
12 </div>
13
```

- ✓ Neste exemplo, o layout se adapta automaticamente: em ecrãs pequenos, ambos os conteúdos ocupam toda a largura (`col-sm-12`), mas em ecrãs médios e maiores, o conteúdo principal ocupa 8 colunas (`col-md-8`) e a barra lateral ocupa 4 colunas (`col-md-4`).

❖ **Conclusão:**

- ✓ O Bootstrap facilita a criação de páginas responsivas ao fornecer ferramentas e componentes que se ajustam automaticamente a diferentes tamanhos de ecrã. Ele é amplamente utilizado para garantir que sites e aplicações ofereçam uma boa experiência de utilizador, independentemente do dispositivo usado para acessá-los.