



Estácio

CadastroEE

Matheus José Ribeiro de Moura 2023 0713 6158

Polo Rua Tereza - Petrópolis - RJ

Vamos integrar sistemas – Turma 9001 – 2024.3 FLEX

Objetivo da Prática

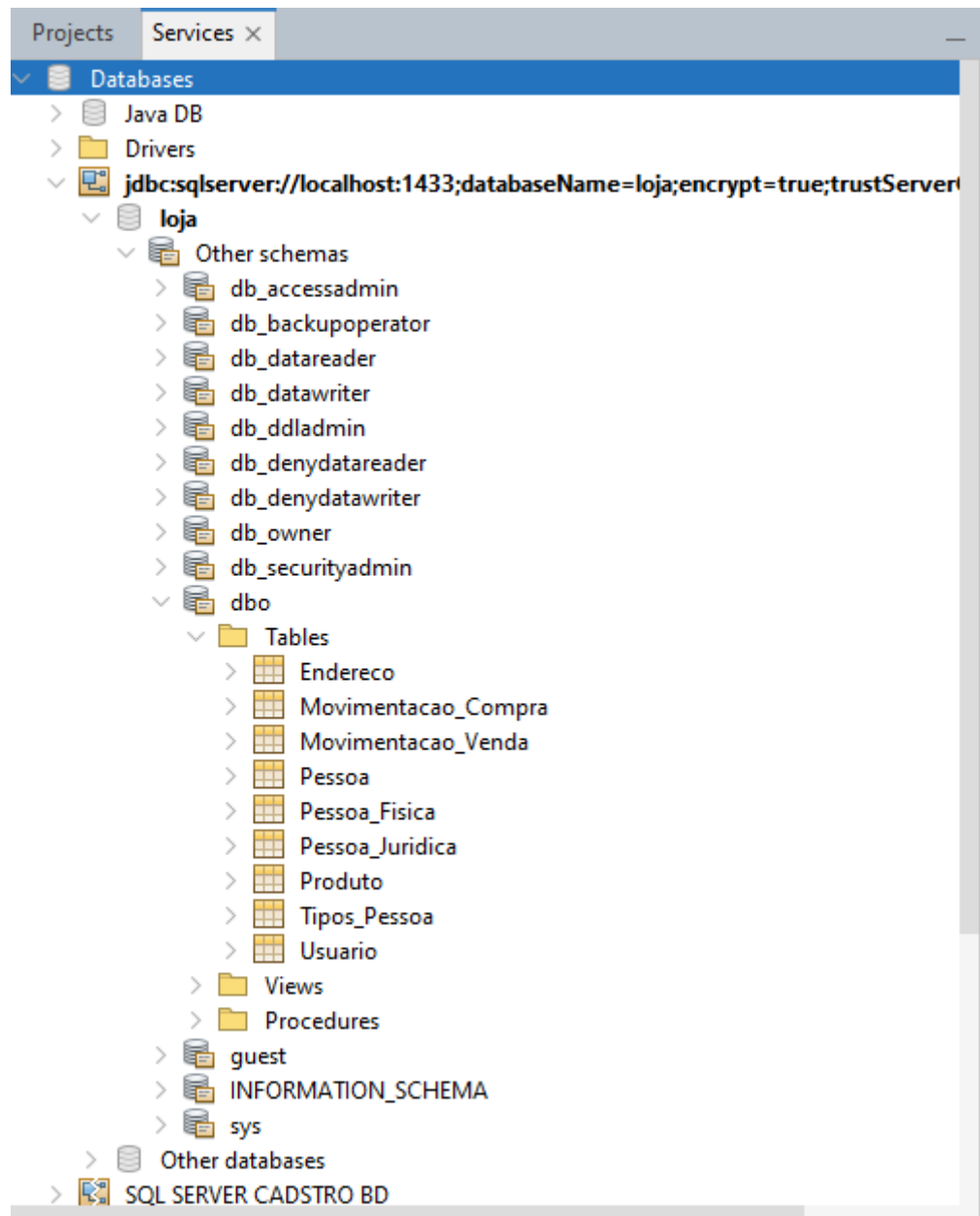
A prática tem como objetivo implementar a persistência com base JPA, implementar as regras de negócio na plataforma JEE, através de EJB, implementar sistema cadastral Web com base em Servlets e JSP, utilizar a biblioteca Bootstrap para melhoria do design. No final do exercício, será criada uma aplicação Java Web, com entrada e exibição de dados e lidando com conexões reais.

Respositório GIT: <https://github.com/MatheusJRM/CadastroEE.git>

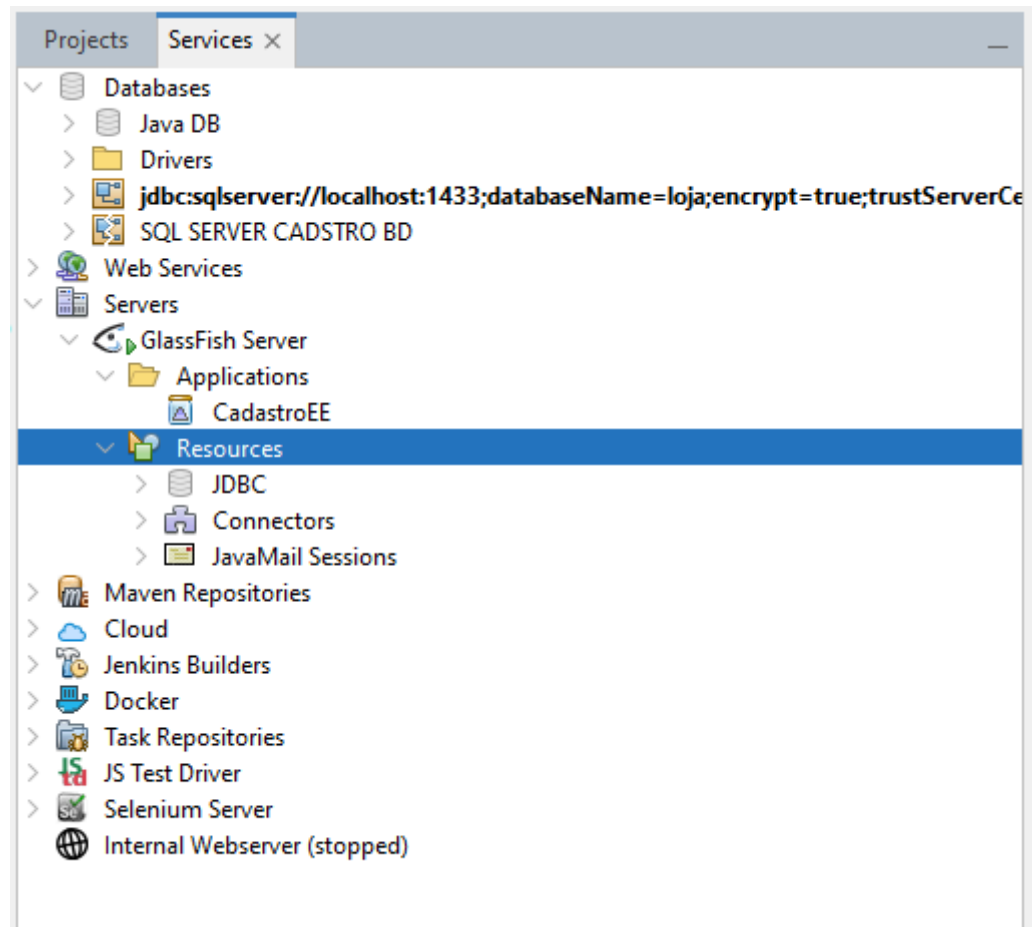
1º Procedimento | Camadas de Persistência e Controle

- Conexão criada com banco SQL SERVER com o drive

mssql-jdbc-12.8.1.jre8.jar



- Criação do Servidor do Glassfish

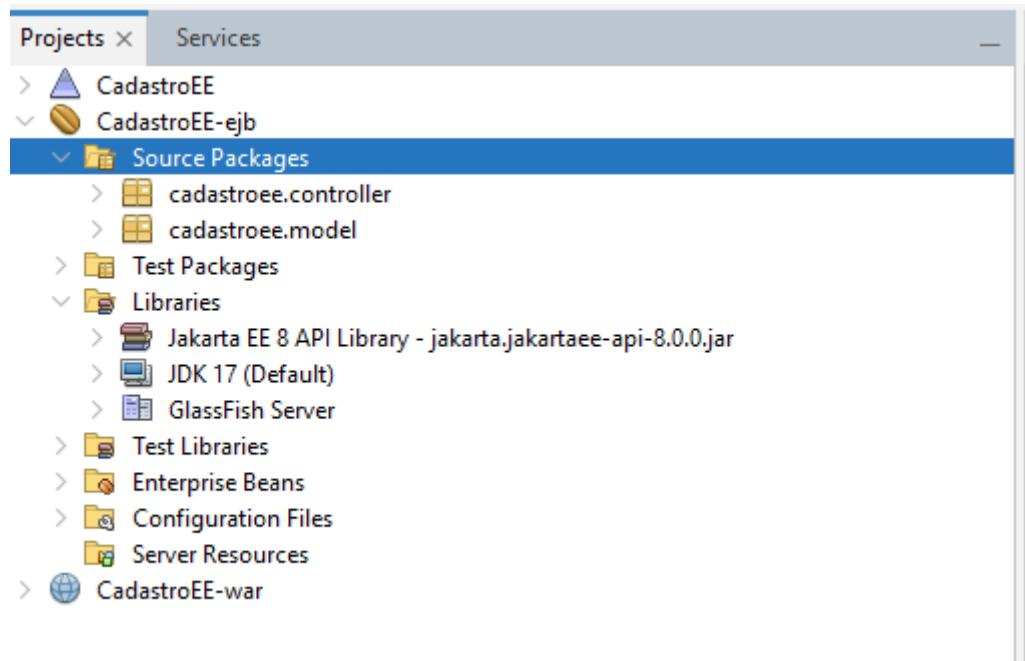


- Criação da Connection Pool e Resources no console admin do Glassfish

JDBC Resources <small>JDBC resources provide applications with a means to connect to a database.</small>						
Resources (4) <small>New... Delete Enable Disable</small>						
Select	JNDI Name	Logical JNDI Name	Enabled	Connection Pool	Description	
<input type="checkbox"/>	jdbc/_TimerPool		✓	TimerPool		
<input type="checkbox"/>	jdbc/_default	java:comp/DefaultDataSource	✓	DerbyPool		
<input type="checkbox"/>	jdbc/loja		✓	SQLServerPool		
<input type="checkbox"/>	jdbc/sample		✓	SamplePool		

JDBC Connection Pools <small>To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection.</small>			
Pools (4) <small>New... Delete</small>			
Select	Pool Name	Resource Type	Classname
<input type="checkbox"/>	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource
<input type="checkbox"/>	SQLServerPool	javax.sql.DataSource	com.microsoft.sqlserver.jdbc.SQLServerDataSource
<input type="checkbox"/>	SamplePool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource
<input type="checkbox"/>	_TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource

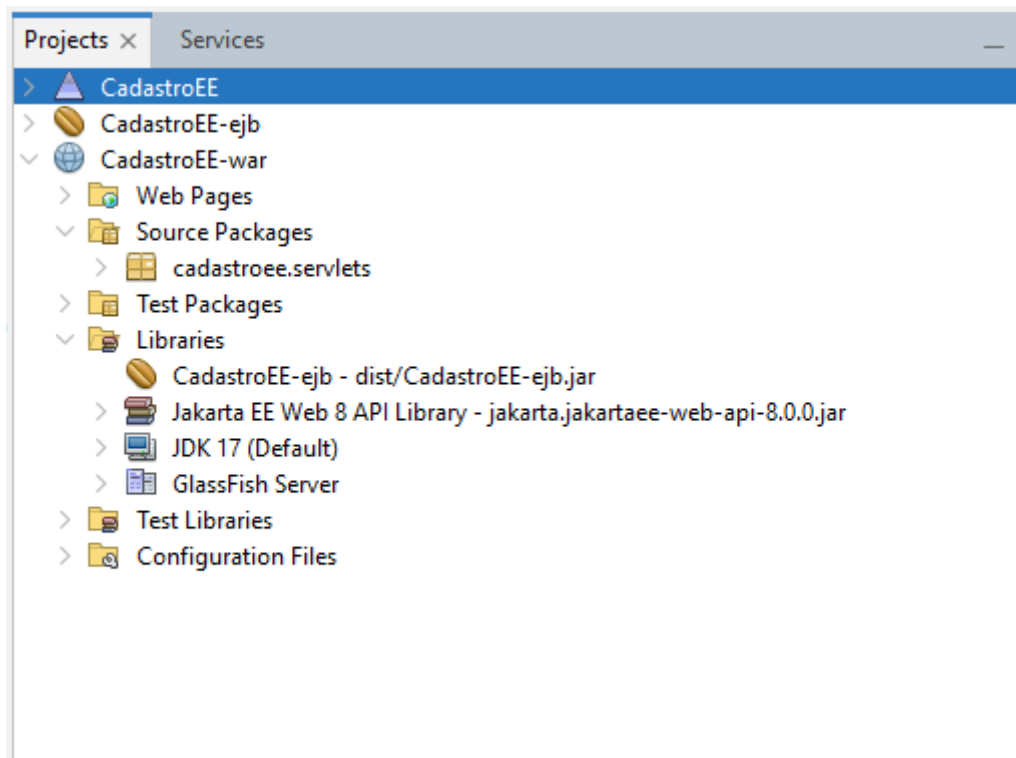
- Estrutura de pastas do projeto CadastroEE-ejb



- Arquivo *persistence.xml* alterado

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="CadastroEE-ejbPU" transaction-type="JTA">
    <jta-data-source>jdbc/loja</jta-data-source>
    <exclude-unlisted-classes>>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

- Estrutura do projeto CadastroEE-war



- Servlet produto para exibição dos dados

```
package cadastroee.servlets;

import cadastroee.controller.ProdutoFacadeLocal;
import cadastroee.model.Produto;
import jakarta.ejb.EJB;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.util.List;

/**
 *
 * @author mathe
```

```

*/
public class ServletProduto extends HttpServlet {

    @EJB
    private ProdutoFacadeLocal facade;

    protected void processRequest(HttpServletRequest
request, HttpServletResponse response)
        throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            List<Produto> produtos = facade.findAll();

            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Lista de
Produtos</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Lista de Produtos</h1>");
            out.println("<ul>");

            for (Produto produto : produtos) {
                out.println("<li>" + produto.getNome() + "
- " + produto.getPrecoVenda() + "</li>");
            }

            out.println("</ul>");
            out.println("</body>");
            out.println("</html>");
        }
    }

// <editor-fold defaultstate="collapsed" desc="HttpServlet
methods. Click on the + sign on the left to edit the
code.">

```

```

/**
 * Handles the HTTP <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific
error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request,
HttpServletRequest response)
    throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific
error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request,
HttpServletRequest response)
    throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override

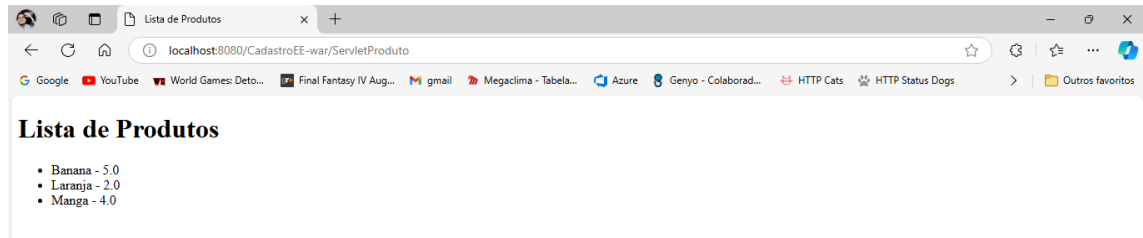
```

```
public String getServletInfo() {  
    return "Short description";  
} // </editor-fold>  
  
}
```

- Arquivo *web.xml* alterado

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app version="4.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">  
  
    <servlet>  
        <servlet-name>ServletProdutoFC</servlet-name>  
        <servlet-class>cadastroee.servlets.ServletProdutoFC</servlet-class>  
    </servlet>  
  
    <servlet-mapping>  
        <servlet-name>ServletProdutoFC</servlet-name>  
        <url-pattern>/produtos</url-pattern>  
    </servlet-mapping>  
  
    <session-config>  
        <session-timeout>30</session-timeout>  
    </session-config>  
</web-app>
```


- **Execução inicial do projeto**



a) Como é organizado um projeto corporativo no NetBeans?

R: Um projeto corporativo no NetBeans é estruturado em módulos, como módulos de apresentação (front-end), lógica de negócios (EJB) e acesso a dados (JPA), permitindo uma separação clara de responsabilidades.

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

R: JPA (Java Persistence API) é utilizada para gerenciar a persistência de dados em bancos de dados, enquanto EJB (Enterprise JavaBeans) fornece uma arquitetura para construir componentes de negócios robustos, permitindo transações, segurança e escalabilidade em aplicativos web.

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

R: O NetBeans oferece ferramentas integradas, como assistentes para criação de entidades JPA e EJB, além de suporte a depuração e testes, que agilizam o desenvolvimento e reduzem erros.

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

R: Servlets são componentes Java que processam requisições HTTP e gerenciam respostas em aplicações web. O NetBeans facilita a criação de Servlets com templates, assistentes e integração com o servidor de aplicações.

e) Como é feita a comunicação entre os Serlvets e os Session Beans do pool de EJBs?

R: A comunicação é realizada através da injeção de dependência ou busca de EJBs via JNDI (Java Naming and Directory Interface) dentro dos Servlets, permitindo que os Servlets chamem métodos dos Session Beans para executar a lógica de negócios.

2º Procedimento | Interface Cadastral com Servlet e JPSs

- **ServletProdutoFC**

```
package cadastroee.servlets;

import java.io.IOException;
import jakarta.ejb.EJB;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import cadastroee.controller.ProdutoFacadeLocal;
import cadastroee.model.Produto;

@WebServlet(name = "ServletProdutoFC", urlPatterns = {"/produtos"})
public class ServletProdutoFC extends HttpServlet {

    @EJB
    private ProdutoFacadeLocal facade; // Referência ao EJB ProdutoFacadeLocal

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String acao = request.getParameter("acao"); // Captura o parâmetro 'acao'
        if (acao != null) {
            acao = acao.trim(); // Remove espaços em branco
        }
        System.out.println("Ação recebida: " + acao);
        String destino = null;

        if (acao == null) {
            acao = "listar";
        }

        switch (acao) {
```

```

        case "listar" -> {
            request.setAttribute("produtos", facade.findAll());
            destino = "ProdutoLista.jsp";
        }
        case "formAlterar" -> {
            int idAlterar = Integer.parseInt(request.getParameter("id"));
            Produto produtoAlterar = facade.find(idAlterar);
            request.setAttribute("produto", produtoAlterar);
            destino = "ProdutoDados.jsp";
        }
        case "excluir" -> {
            int idExcluir = Integer.parseInt(request.getParameter("id"));
            Produto produtoExcluir = facade.find(idExcluir);
            facade.remove(produtoExcluir);
            request.setAttribute("produtos", facade.findAll());
            destino = "ProdutoLista.jsp";
        }
        case "alterar" -> {
            int idAlterarDados = Integer.parseInt(request.getParameter("id"));
            Produto produtoAlterarDados = facade.find(idAlterarDados);
            produtoAlterarDados.setNome(request.getParameter("nome"));
            produtoAlterarDados.setPrecoVenda(Float.valueOf(request.getParameter("preco")));
            produtoAlterarDados.setQuantidadeEstoque(Integer.parseInt(request.getParameter("quantidade")));
            facade.edit(produtoAlterarDados);
            request.setAttribute("produtos", facade.findAll());
            destino = "ProdutoLista.jsp";
        }
        case "incluir" -> {
            Produto novoProduto = new Produto();
            novoProduto.setNome(request.getParameter("nome"));
            novoProduto.setPrecoVenda(Float.valueOf(request.getParameter("preco")));
            novoProduto.setQuantidadeEstoque(Integer.parseInt(request.getParameter("quantidade")));
            facade.create(novoProduto);
            request.setAttribute("produtos", facade.findAll());
            destino = "ProdutoLista.jsp";
        }
        case "formIncluir" ->
            destino = "ProdutoDados.jsp";
        default -> {
            request.setAttribute("produtos", facade.findAll());
            destino = "ProdutoLista.jsp";
        }
    }

    RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
    dispatcher.forward(request, response);
}

```

```

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

@Override
public String getServletInfo() {
    return "Servlet para gerenciamento de produtos";
}
}

```

- **ProdutoLista.jsp**

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="cadastroee.model.Produto" %>

<%
    List<Produto> produtos = (List<Produto>) request.getAttribute("produtos");
%>

<!DOCTYPE html>
<html lang="pt-BR">
    <head>
        <meta charset="UTF-8">
        <title>Lista de Produtos</title>
        <link rel="stylesheet" href="styles.css">
    </head>
    <body>
        <h1>Lista de Produtos</h1>
        <a href="produtos?acao=formIncluir">Incluir Novo Produto</a>
        <table border="1">

```

```

<thead>
  <tr>
    <th>ID</th>
    <th>Nome</th>
    <th>Preço</th>
    <th>Quantidade em Estoque</th>
    <th>Ações</th>
  </tr>
</thead>
<tbody>
  <%
    if (produtos != null && !produtos.isEmpty()) {
      for (Produto produto : produtos) {
  %>
    <tr>
      <td><%= produto.getId()%></td>
      <td><%= produto.getNome()%></td>
      <td><%= produto.getPrecoVenda()%></td>
      <td><%= produto.getQuantidadeEstoque()%></td>
      <td>
        <a href="produtos?acao=formAlterar&id=<%=
produto.getId()%>">Alterar</a>
        <a href="produtos?acao=excluir&id=<%= produto.getId()%>"
onclick="return confirm('Tem certeza que deseja excluir este
produto?');">Excluir</a>
      </td>
    </tr>
  <%
    }
  } else {
  %>
    <tr>
      <td colspan="5">Nenhum produto encontrado.</td>
    </tr>
  <%
    }
  %>

```

```

        </tbody>
    </table>
</body>
</html>

```

- **ProdutoDados.jsp**

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>Cadastro de Produto</title>
        <link rel="stylesheet" type="text/css" href="styles.css">
    </head>
    <body>
        <h1><c:choose>
            <c:when test="${not empty produto}">
                Alterar Produto
            </c:when>
            <c:otherwise>
                Incluir Novo Produto
            </c:otherwise>
        </c:choose></h1>

        <form action="produtos" method="post">
            <input type="hidden" name="acao" value="<c:choose>
                <c:when test="${not empty produto}">alterar</c:when>
                <c:otherwise>incluir</c:otherwise>
            </c:choose>"/>

            <c:if test="${not empty produto}">
                <input type="hidden" name="id" value="${produto.id}"/>
            </c:if>

            <label for="nome">Nome:</label>
            <input type="text" id="nome" name="nome" value="${not empty

```

```
produto ? produto.nome : "" required/>

<label for="quantidade">Quantidade:</label>
<input type="number" id="quantidade" name="quantidade"
value="{not empty produto ? produto.quantidadeEstoque : ""} required/>

<label for="preco">Preço de Venda:</label>
<input type="number" step="0.01" id="preco" name="preco"
value="{not empty produto ? produto.precoVenda : ""} required/>

<button type="submit">
  <c:choose>
    <c:when test="{not empty produto}">Alterar Produto</c:when>
    <c:otherwise>Incluir Produto</c:otherwise>
  </c:choose>
</button>
</form>

<br/>
<a href="produtos">Voltar para a lista de produtos</a>
</body>
</html>
```

- **Resultado da execução dos códigos**

- **Listar**

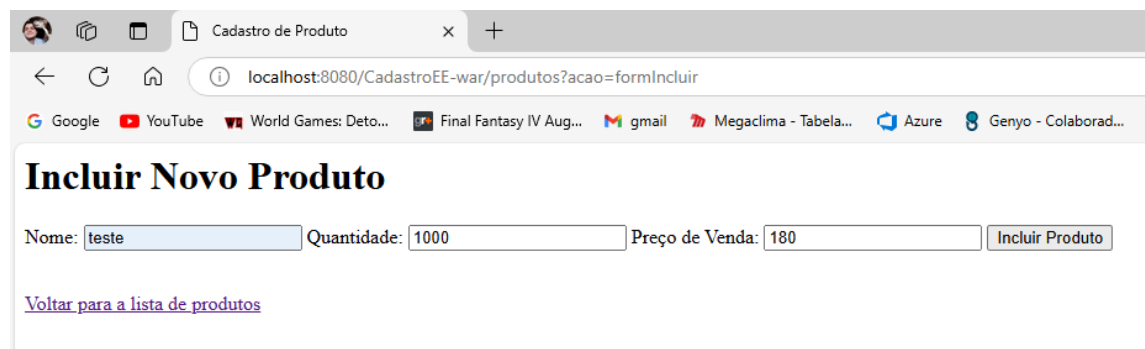


Lista de Produtos

[Incluir Novo Produto](#)

ID	Nome	Preço	Quantidade em Estoque	Ações
2	Laranja	2.0	500	Alterar Excluir
3	Manga	4.0	800	Alterar Excluir
4	Banana	2.25	100	Alterar Excluir

- **FormIncluir**



Incluir Novo Produto

Nome: Quantidade: Preço de Venda:

[Voltar para a lista de produtos](#)

- **Incluir**

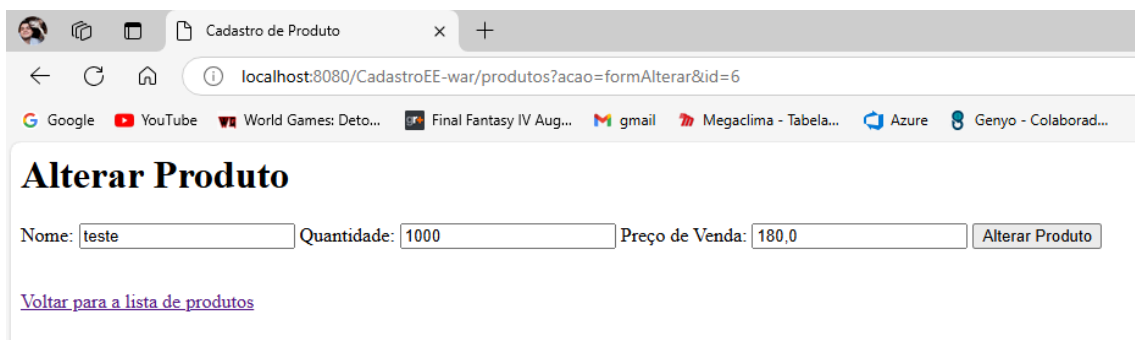


Lista de Produtos

[Incluir Novo Produto](#)

ID	Nome	Preço	Quantidade em Estoque	Ações
2	Laranja	2.0	500	Alterar Excluir
3	Manga	4.0	800	Alterar Excluir
4	Banana	2.25	100	Alterar Excluir
6	teste	180.0	1000	Alterar Excluir

- **AlterarForm**

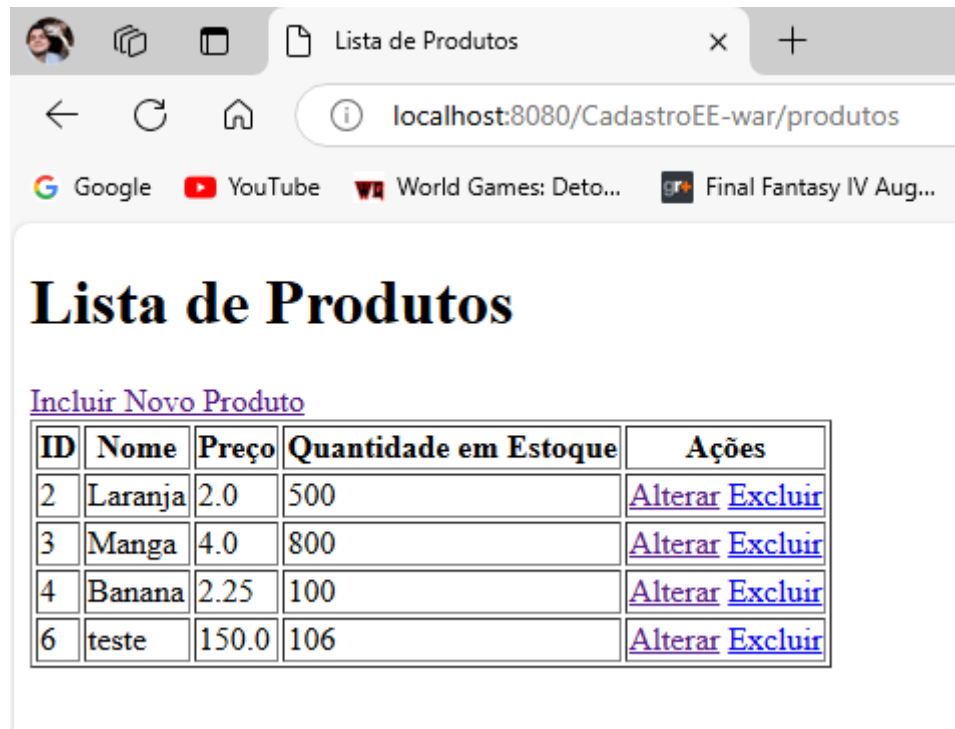


Alterar Produto

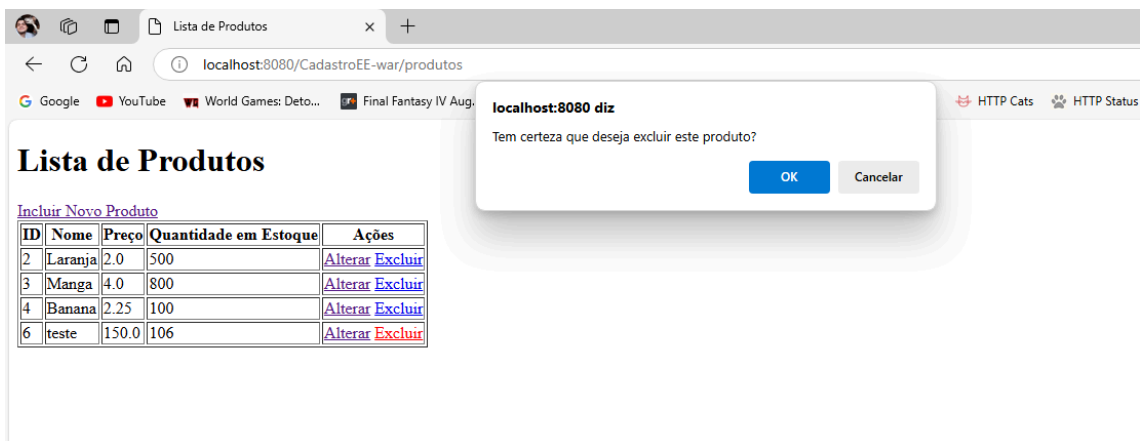
Nome: Quantidade: Preço de Venda:

[Voltar para a lista de produtos](#)

- **Alterar**



- **Excluir**





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

R: O padrão Front Controller centraliza o processamento de requisições em um único ponto de entrada, geralmente um servlet. Em um aplicativo Web Java na arquitetura MVC, o Front Controller recebe todas as requisições, as direciona para os controladores apropriados e retorna a resposta ao cliente, facilitando a separação de responsabilidades e a manutenção do código.

- b) Quais as diferenças e semelhanças entre Servlets e JSPs?

R: Ambos são componentes da tecnologia Java EE para criar aplicações web e podem interagir com o cliente através de requisições e respostas HTTP. A diferença é que os Servlets são classes Java que processam requisições e geram respostas, enquanto JSPs (JavaServer Pages) são arquivos que permitem a inclusão de código Java em HTML, facilitando a criação de interfaces de usuário. JSPs são compilados em Servlets pelo servidor.

- c) 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?

R: O redirecionamento (`response.sendRedirect`) envia uma nova requisição ao cliente, mudando a URL no navegador. O método forward

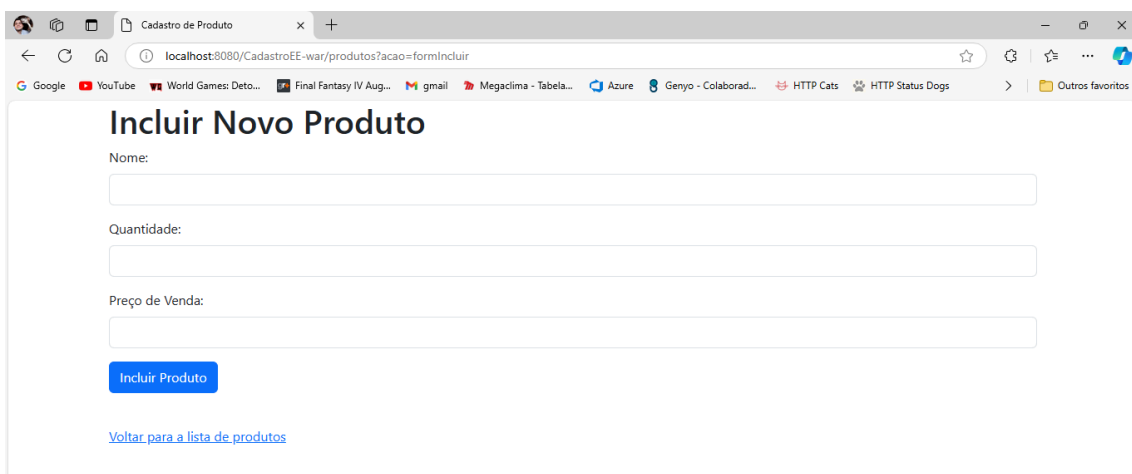
(requestDispatcher.forward) mantém a URL original e encaminha a requisição para outro recurso no servidor, permitindo compartilhar atributos. Os parâmetros são dados enviados na URL ou no corpo da requisição, enquanto atributos são objetos armazenados no contexto da requisição, que podem ser usados para passar informações entre servlets e JSPs durante o processamento da mesma requisição.

3º Procedimento | Melhorando o Design da Interface

- Listagem estilizada



- Página de inclusão/alteração estilizada



Alterar Produto

Nome:

Quantidade:

Preço de Venda:

[Alterar Produto](#)

[Voltar para a lista de produtos](#)

a) Como o framework Bootstrap é utilizado?

R: Para usar o Bootstrap, você deve incluir os arquivos CSS e JavaScript em seu projeto (via CDN ou download). Em seguida, você pode aplicar classes pré-definidas para criar layouts, botões, formulários e outros componentes de forma rápida e eficiente.

b) Por que o Bootstrap garante a independência estrutural do HTML?

R: O Bootstrap separa a estrutura (HTML) da apresentação (CSS), permitindo que o desenvolvedor altere o estilo sem modificar a estrutura do conteúdo, o que facilita a manutenção e a reutilização do código.

c) Qual a relação entre o Bootstrap e a responsividade da página?

R: O Bootstrap utiliza um sistema de grid flexível e classes CSS que permitem que os elementos se ajustem automaticamente a diferentes tamanhos de tela, garantindo que as páginas sejam responsivas.

Conclusão

A realização de um projeto JavaEE utilizando o GlassFish como servidor de aplicação proporcionou uma compreensão abrangente e prática sobre a integração entre as diversas camadas de uma aplicação web. A comunicação com o banco de dados foi facilitada pelo uso do JPA (Java Persistence API), que permite a manipulação de dados de forma eficiente e orientada a objetos, abstraindo a complexidade das operações de banco de dados.

Além disso, a implementação de Servlets possibilitou a criação de uma lógica de controle robusta, gerenciando as requisições e respostas entre o front-end e o back-end de maneira eficiente. Essa interação é fundamental para a dinâmica da aplicação, permitindo que as informações sejam processadas e apresentadas ao usuário de forma fluida.

Por fim, a utilização do Bootstrap no front-end não apenas garantiu um design responsivo e atraente, mas também facilitou a integração com o back-end, permitindo que os dados manipulados pelos Servlets fossem apresentados de maneira clara e acessível. Essa experiência prática consolidou o entendimento sobre a arquitetura de aplicações web, ressaltando a importância da comunicação entre as camadas e a eficiência na construção de sistemas escaláveis e manuteníveis. Em suma, o projeto não apenas aprimorou habilidades técnicas,

mas também ofereceu uma visão holística do desenvolvimento de software, essencial para a construção de aplicações modernas.