



FACULDADE ESTÁCIO DE SÁ

CURSO: DESENVOLVIMENTO FULL STACK

3º SEMESTRE – MATRÍCULA 202302595341

Repositório GitHub - [alaimalmeida/CadastroEE \(github.com\)](https://github.com/alaimalmeida/CadastroEE)

ALAIM ALMEIDA DE OLIVEIRA

Vamos Integrar Sistemas

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

Salvador – BA

2024

1. Introdução

A implementação de sistemas cadastrais é essencial para a gestão eficiente de dados em diversas organizações. A utilização de tecnologias robustas e bem estabelecidas no mercado, como Java Enterprise Edition (JEE), Servlets, Java Persistence API (JPA) e Bootstrap, permite criar sistemas escaláveis, eficientes e com interfaces amigáveis.

Grande parte das empresas usam sistemas de cadastro, sejam empresas de pequeno porte e empresas de grande porte, indústrias e etc.

2. Estrutura

O sistema está estruturado da seguinte forma:

- 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.

3. Objetivo

O objetivo deste projeto é desenvolver um sistema cadastral web utilizando as tecnologias JEE, JPA, EJBs, Servlets, JSPs e Bootstrap, visando criar uma aplicação que seja:

- Robusta: Implementando lógica de negócios segura e eficiente.
- Escalável: Utilizando JPA para gerenciamento de persistência de dados.
- Amigável ao usuário: Através de uma interface moderna e responsiva com Bootstrap.
- Modular: Facilitando a manutenção e expansão futura do sistema.

Esse projeto foi utilizado o JDK e IDE NetBeans, juntamente com o banco de dados SQL Server com o Management Studio, para poder fazer todo armazenamento de dados cadastrado pelo sistema.

No primeiro procedimento, fizemos toda a configuração do banco de dados, inserindo o drive **mssql-jdbc-12.2.0.jre8.jar** e configurando a conexão utilizando o **user: loja, password: loja, dataBase: Loja** e inserindo o JDBC URL

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

Ficando dessa forma logo abaixo:

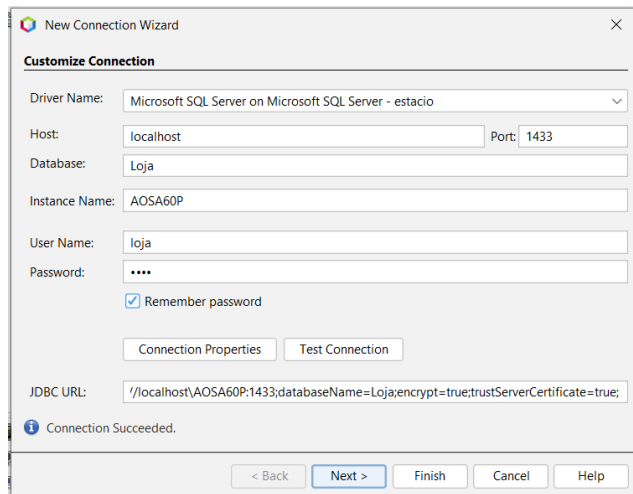


Imagem 01 – Conexão o banco

Depois da configuração com o banco de dados, foi a vez de configurar o servidor GlassFish Server 7.0.9 e acrescentando a biblioteca ao mesmo **mssql-jdbc-12.2.0.jre8.jar**. Segue abaixo a janela de sua configuração:

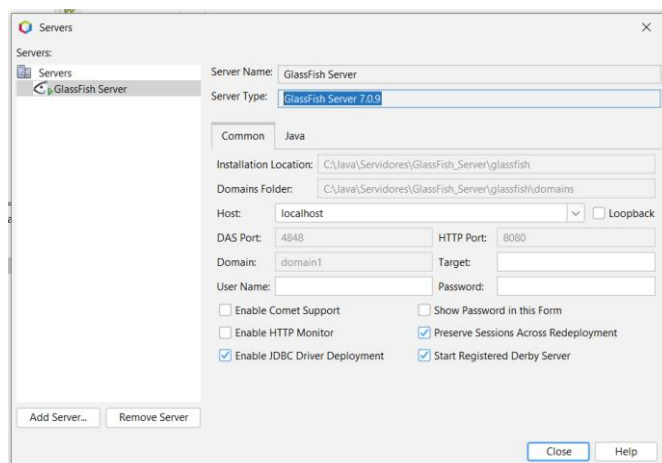


Imagem 02 – Configuração GlassFish 7.0.9

Foi criado um projeto do tipo Ant...Java Enterprise..Enterprise Application como mostra na figura abaixo:

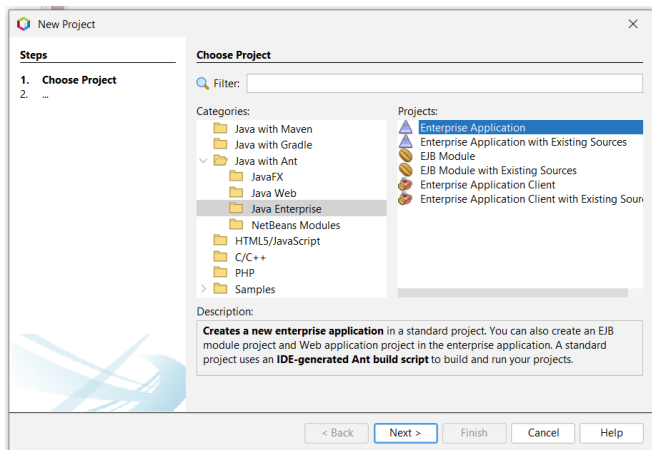


Imagem 03 – Criação do projeto

Logo em seguida, 3 projetos foram gerados, o principal que encapsula o arquivo EAR, tendo os outros dois, CadastroEE-ejb e CadastroEE-war como projetos dependentes relacionados aos elementos JPA, JEE e Web.

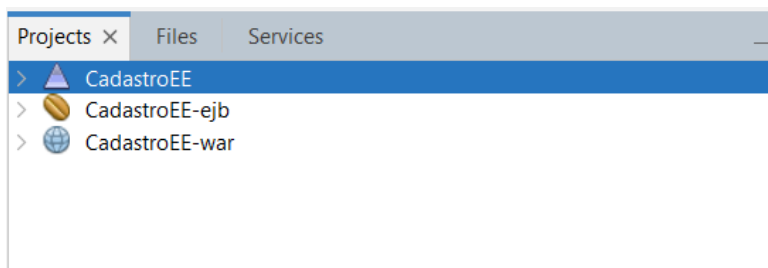


Imagem 04 – Os três projetos criados

Próximo passo, foi definido as camadas de persistências e controle no projeto CadastroEE-ejb, criando as entidades JPA através de New Entity Classes from Database, selecionando jdbc/loja como Data Source e selecionando todas as tabelas que compõem o banco de dados criado logo no início:

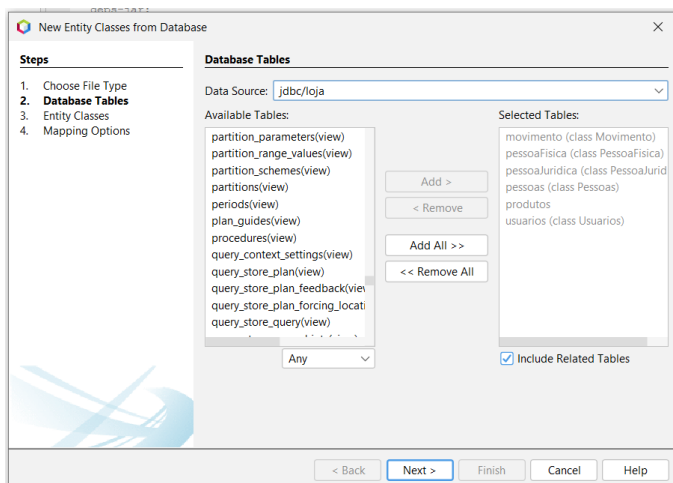


Imagem 05 – Seleccionando as tabelas

Depois de ter configurado as tabelas de persistência, foi criado o pacote cadastroee.model marcando a opção de criação do arquivo persistence.xml.

Adicionando a opção de New Session Beans for Entity Classes e selecionando todas as entidades, para a geração das interfaces local além de definir o pacote cadastroee.controller.

Foram criadas os Session Beans com o sufixo Facade e como as interfaces com o sufixo FacadeLocal e adicionando a sua biblioteca o Jakarta EE 8 API ao projeto CadatroEE-ejb.

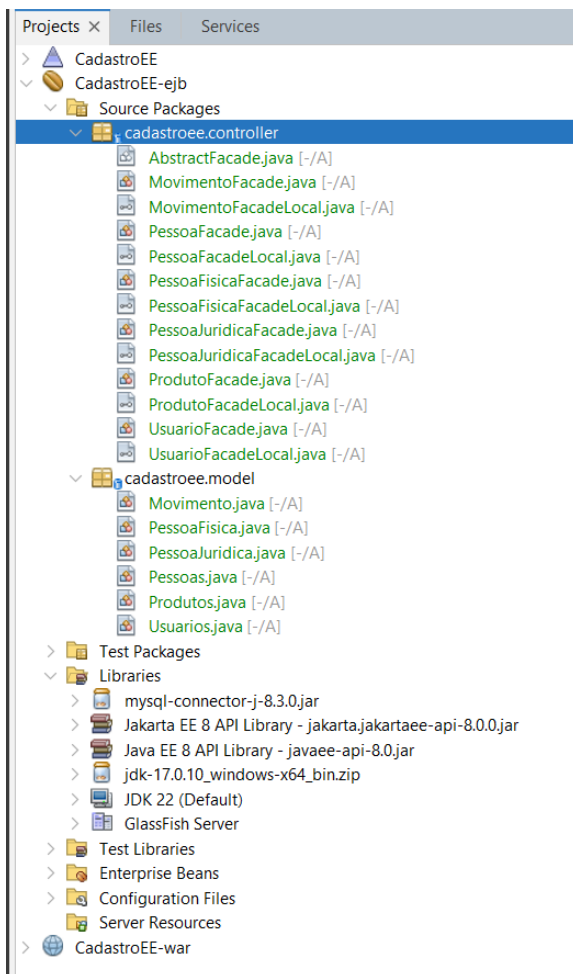


Imagem 05 – Estrutura do pacote cadastroee.controller, cadastroee.model e a biblioteca

Depois de ter feito as configurações acima, todas as importações foram trocadas de javax para jakarta nos arquivos CadastroEE-ejb.

```
6
7  import cadastroee.model.Produtos;
8  import jakarta.ejb.Stateless;
9  import jakarta.persistence.EntityManager;
0  import jakarta.persistence.PersistenceContext;
1
```

Imagem 06 – Troca da importação de javax para jakarta

Na página de produto, o atributo precoVenda foi modificado de BigDecimal para Float.

```
54 @Column(name = "precoVenda")
55 private float precoVenda;
56 @OneToMany(cascade = CascadeType.ALL, mappedBy = "idProduto")
57 private Collection<Movimento> movimentoCollection;
58
59 public Produtos() {
60 }
61
62 public Produtos(Integer idProduto) {...3 lines }
63
64 public Produtos(Integer idProduto, String nome) {...4 lines }
65
66 public Integer getIdProduto() {...3 lines }
67
68 public void setIdProduto(Integer idProduto) {...3 lines }
69
70 public String getNome() {...3 lines }
71
72 public void setNome(String nome) {...3 lines }
73
74 public Integer getQuantidade() {...3 lines }
75
76 public void setQuantidade(Integer quantidade) {...3 lines }
77
78 public float getPrecoVenda() {
79     return precoVenda;
80 }
81
82 public void setPrecoVenda(float precoVenda) {
83     this.precoVenda = precoVenda;
84 }
```

Imagem 07 – troca do atributo precoVenda de BigDecimal para Float

No arquivo persistence.xml, foi feita uma modificação, pois foi acrescentado todas as tabelas do projeto e sua conexão ao banco de dados completo.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="3.0" xmlns="https://jakarta.ee/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
4   <persistence-unit name="CadastroEE-ejbPU" transaction-type="JTA">
5     <jta-data-source>jdbc/loja</jta-data-source>
6     <class>com.cadastroee.Movimento</class>
7     <class>com.cadastroee.PessoaFisica</class>
8     <class>com.cadastroee.PessoaJuridica</class>
9     <class>com.cadastroee.Pessoas</class>
10    <class>com.cadastroee.Produtos</class>
11    <class>com.cadastroee.Usuarios</class>
12    <properties>
13      <property name="jakarta.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
14      <property name="jakarta.persistence.jdbc.url" value="jdbc:sqlserver://localhost:1433;databaseName=Loja;encrypt=true;trustServerCertificate=true"/>
15      <property name="jakarta.persistence.jdbc.user" value="sa"/>
16      <property name="jakarta.persistence.jdbc.password" value="S777"/>
17      <property name="jakarta.persistence.schema-generation.database.action" value="create"/>
18      <property name="jakarta.persistence.schema-generation.database.action" value="create"/>
19    </properties>
20  </persistence-unit>
21 </persistence>
```

Imagem 08 – Conexão com o banco

No projeto CadastroEE-war, foi criado uma pasta com o nome Servlet com o arquivo chamado ServletProduto, ficando dentro do pacote cadastroee.servlets, seguido da opção marcada Add information to deployment descriptor.

Com isso, foi adicionado a referência para a interface EJB

```
1  *
2  * Click https://ghhost/SystemFileSystem/Templates/license-default.txt to change this license
3  * Click https://ghhost/SystemFileSystem/Templates/JSP\_Servlet/Servlet.java to edit this template
4  */
5  package cadastroee.servlets;
6
7  import cadastroee.controller.ProdutoFacadeLocal;
8  import cadastroee.model.Produto;
9  import jakarta.ejb.EJB;
10 import java.io.IOException;
11 import java.io.PrintWriter;
12 import jakarta.servlet.ServletException;
13 import jakarta.servlet.http.HttpServlet;
14 import jakarta.servlet.http.HttpServletRequest;
15 import jakarta.servlet.http.HttpServletResponse;
16 import java.text.DecimalFormat;
17 import java.util.List;
18
19 /**
20  *
21  * @author Alain
22  */
23
24 public class ServletProduto extends HttpServlet {
25     @EJB
26     private ProdutoFacadeLocal facade;
27
28     /**
29      * Processes requests for both HTTP GET and POST methods.
30      *
31      * @param request HttpServletRequest request
32      * @param response HttpServletResponse response
33      * @throws ServletException if a Servlet-specific error occurs
34      * @throws IOException if an I/O error occurs
35      */
36 }
```

Imagem 09 – Interface do EJB

Logo em seguida, foi adicionado a biblioteca Jakarta EE Web 8 API ao projeto CadastroEE-war

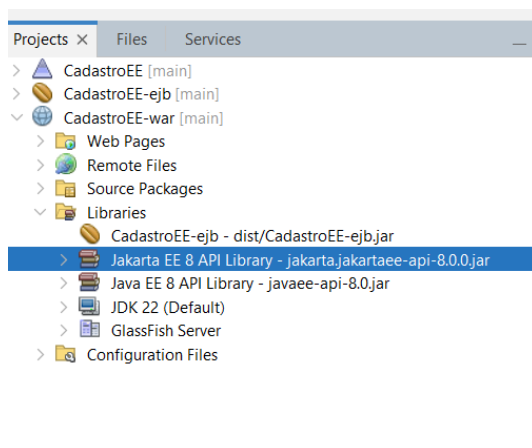


Imagem 10 – Biblioteca Jakarta EE Web 8 API

Com todas as importações de pacotes javax para jakarta, em todos os arquivos do projeto CadastroEE-war, podemos executar com Run ou Deploy no projeto principal (CadastroEE). A demonstração no navegador fica da seguinte forma:

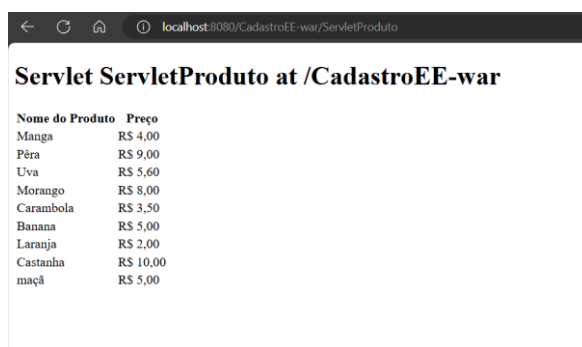


Imagem 11 – Resultado

Depois de todo procedimento, foi criado um Servlet com o nome ServletProdutoFC, no projeto CadastroEE-war, utilizando o padrão Front Controller e adicionando uma referência para ProdutoFacadeLocal, utilizando o nome facade para o atributo.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/JSP_Servlet/Servlet.java to edit this template
4  */
5  package cadastroee.servlets;
6
7  import cadastroee.controller.ProdutoFacadeLocal;
8  import cadastroee.model.Produtos;
9  import jakarta.ejb.EJB;
10 import java.io.IOException;
11 import java.io.PrintWriter;
12 import jakarta.servlet.ServletException;
13 import jakarta.servlet.http.HttpServlet;
14 import jakarta.servlet.http.HttpServletRequest;
15 import jakarta.servlet.http.HttpServletResponse;
16 import java.text.DecimalFormat;
17 import java.util.List;
18
19 /**
20 *
21 * @author Alain
22 */
23
24 public class ServletProduto extends HttpServlet {
25     @EJB
26     private ProdutoFacadeLocal facade;
27
28     /**
29     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
30     * methods.
31     *
32     * @param request <code>HttpServletRequest</code> request
33     * @param response <code>HttpServletResponse</code> response
34     * @throws ServletException if a <code>servlet-specific</code> error occurs
35     * @throws IOException if an I/O error occurs
36     */
37 }
```

Imagem 12 – ServletProdutoFC

```
38
39 @Override
40 protected void doGet(HttpServletRequest request, HttpServletResponse response)
41     throws ServletException, IOException {
42     response.setContentType("text/html; charset=UTF-8");
43     request.setCharacterEncoding("UTF-8");
44
45     List<Produtos> listaDeProdutos = facade.findAll();
46     String htmlResponse = buildHtmlResponse(listaDeProdutos);
47
48     try (PrintWriter out = response.getWriter()) {
49         out.println(htmlResponse);
50     }
51
52     private String buildHtmlResponse(List<Produtos> produtos) {
53         StringBuilder htmlBuilder = new StringBuilder();
54         DecimalFormat df = new DecimalFormat("#,##0.00");
55
56         htmlBuilder.append("<!DOCTYPE html>")
57             .append("<html>")
58             .append("<head>")
59             .append("<meta charset='UTF-8'>")
60             .append("<title>Listagem de Produtos</title>")
61             .append("</head>")
62             .append("<body>")
63             .append("<h1>Servlet ServletProduto at /CadastroEE-war</h1>")
64             .append("<table >")
65             .append("<tr><th>Nome do Produto</th><th>Preço</th></tr>");
66
67         for (Produtos produto : produtos) {
68             htmlBuilder.append("<tr><td>")
69                 .append(produto.getNome())
70                 .append("</td><td>R$ ")
71                 .append(df.format(produto.getPrecoVenda()))
72                 .append("</td></tr>");
73 }
```

Imagem 13 - ServletProdutoFC

```
74
75     htmlBuilder.append("</table>")
76         .append("</body>")
77         .append("</html>");
78
79     return htmlBuilder.toString();
80
81
82 @Override
83 public String getServletInfo() {
84     return "Servlet para listagem de produtos";
85 }
86
87
88 }
```

Imagem 14 - ServletProdutoFC

Ainda nesse procedimento, foram criados dois arquivos para poder estruturar toda a parte html, o layout, são eles ProdutoDados.jsp e ProdutoLista.jsp. Segue abaixo toda a estrutura do ProdutoLista.jsp:


```

1 Document : ProdutoDados
2 Created on : May 26, 2024, 7:48:17 PM
3 Author : Alain
4
5 -->
6
7 <%@page import="cadastroee.model.Produto"%>
8 <%@page import="java.text.DecimalFormat"%>
9 <%@page import="cadastroee.model.Produto"%>
10 <%@page contentType="text/html" pageEncoding="UTF-8"%>
11 <!DOCTYPE html>
12 <html>
13 <head>
14 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15 <title>Cadastro de Produtos</title>
16 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-4bw4a8h4bpZf0/R2TPyzX5p36E9+IDWx6JGf8VUyc4e8o84FbW50Unv6mLfE4" crossorigin="anonymous">
17 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-ggbgdd+/hkxVVq/SJmZ8WgDXIAcQv1Wuq14Ezjg7561M4ZAi2ZxUKi6Iq5U" crossorigin="anonymous">
18 </script>
19 </head>
20 <body>
21 <div class="container">
22 <div class="header-section">
23 <div class="text-center">
24 <h1>Cadastro de Produtos</h1>
25 </div>
26 </div>
27 <div class="form-container">
28 <div class="row">
29 <div class="col-md-6">
30 <div class="form-label">
31 <input type="text" name="nome" value="" />
32 </div>
33 <div class="form-label">
34 <input type="text" name="descricao" value="" />
35 </div>
36 <div class="form-label">
37 <input type="text" name="preco" value="" />
38 </div>
39 <div class="form-label">
40 <input type="text" name="qtd" value="" />
41 </div>
42 </div>
43 <div class="text-center">
44 <input type="button" value="Cadastrar" />
45 </div>
46 </div>
47 </div>
48 </body>
49 </html>

```

Imagem 15 – ProdutoDados.jsp

```

36 <div class="form-label">
37 <input type="text" name="nome" value="" />
38 </div>
39 <div class="form-label">
40 <input type="text" name="descricao" value="" />
41 </div>
42 <div class="form-label">
43 <input type="text" name="preco" value="" />
44 </div>
45 <div class="form-label">
46 <input type="text" name="qtd" value="" />
47 </div>
48 </div>
49 <div class="text-center">
50 <input type="button" value="Cadastrar" />
51 </div>
52 </div>
53 </div>
54 </body>
55 </html>

```

Imagem 16 – ProdutoDados.jsp

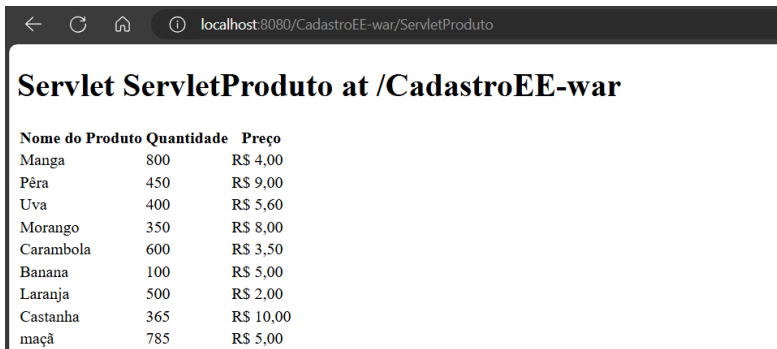
```

68 <input type="hidden" name="acao" value="" />
69 <input type="hidden" name="produtoId" value="" />
70 <div class="form-label">
71 <input type="text" name="nome" value="" />
72 </div>
73 <div class="form-label">
74 <input type="text" name="descricao" value="" />
75 </div>
76 <div class="form-label">
77 <input type="text" name="preco" value="" />
78 </div>
79 <div class="form-label">
80 <input type="text" name="qtd" value="" />
81 </div>
82 </div>
83 <div class="text-center">
84 <input type="button" value="Cadastrar" />
85 </div>
86 </div>
87 </div>
88 </body>
89 </html>

```

Imagem 17 – ProdutoDados.jsp

Depois de ter configurado toda essa parte da Interface Cadastral com Servlet e JSPs, segue abaixo o resultado esperado, puxando as informações do banco de dados as informações dos produtos:



Nome do Produto	Quantidade	Preço
Manga	800	R\$ 4,00
Pêra	450	R\$ 9,00
Uva	400	R\$ 5,60
Morango	350	R\$ 8,00
Carambola	600	R\$ 3,50
Banana	100	R\$ 5,00
Laranja	500	R\$ 2,00
Castanha	365	R\$ 10,00
maçã	785	R\$ 5,00

Imagem 18 – Resultado

Análise e Conclusão

Organização de um Projeto Corporativo no NetBeans

Um projeto empresarial no NetBeans é organizado de forma modular, permitindo uma divisão clara das responsabilidades, o que facilita a manutenção e a escalabilidade do sistema. Em um projeto baseado em Java EE, como o CadastroEE, a estrutura padrão inclui:

- Módulo EJB (Enterprise JavaBeans): Contém componentes de negócio, como session beans, que implementam as regras de negócio e acessam a camada de persistência.
- Módulo WAR (Web Application Archive): Abriga componentes da camada de apresentação, incluindo Servlets, JSPs e recursos estáticos como CSS e JavaScript.
- Módulo EAR (Enterprise Archive): Agrupa os módulos EJB e WAR para serem implantados em um servidor de aplicação Java EE, como o GlassFish.

Função das Tecnologias JPA e EJB na Construção de um Aplicativo Web no Ambiente Java

1. JPA (Java Persistence API):

- Persistência de Dados: JPA é usada para mapear objetos Java para tabelas em um banco de dados relacional, permitindo que operações CRUD (Create, Read, Update, Delete) sejam realizadas de maneira transparente.
- Abstração da Persistência: Simplifica a manipulação de dados sem a necessidade de escrever SQL diretamente, promovendo maior produtividade e facilidade de manutenção do código.

2. EJB (Enterprise JavaBeans):

- Componentes de Negócio: EJBs são usados para implementar a lógica de negócios. Eles podem ser stateful ou stateless, dependendo do tipo de interação necessária com o cliente.
- Gerenciamento de Transações: EJBs gerenciam automaticamente transações, segurança e concorrência, permitindo que os desenvolvedores se concentrem na lógica de negócios.

Aumento da Produtividade com NetBeans ao Usar JPA e EJB

- Assistentes de Código: NetBeans oferece assistentes que facilitam a criação de entidades JPA e session beans EJB, reduzindo a quantidade de código repetitivo que os desenvolvedores precisam escrever.
- Integração com Servidores de Aplicação: Fornece suporte integrado para servidores de aplicação como GlassFish, permitindo testes e depuração diretamente no ambiente de desenvolvimento.
- Facilidade de Configuração: NetBeans auxilia na configuração de unidades de persistência e outros recursos de infraestrutura através de interfaces gráficas, reduzindo a complexidade da configuração manual.

Servlets e Suporte do NetBeans

O que são Servlets:

- Componentes Web: Servlets são classes Java que respondem a solicitações HTTP, permitindo a criação de aplicações web dinâmicas.
- Gerenciamento de Ciclo de Vida: Servlets possuem um ciclo de vida bem definido, gerenciado pelo contêiner de servlets do servidor de aplicação.

Suporte do NetBeans:

- Assistente de Criação: NetBeans inclui assistentes para a criação de Servlets, configurando automaticamente o arquivo de descritor de implantação (web.xml) quando necessário.
- Hot-Deploy: Permite a implantação e atualização de Servlets sem a necessidade de reiniciar o servidor, agilizando o desenvolvimento e testes.

Comunicação entre Servlets e Session Beans do Pool de EJBs

- Injeção de Dependências: Servlets utilizam a anotação @EJB para injetar referências aos session beans, permitindo que os servlets chamem métodos nos EJBs para realizar operações de negócios.
- Chamadas de Métodos: Os servlets capturam as solicitações HTTP, processam os parâmetros e chamam métodos nos EJBs para manipular os dados conforme a lógica de negócios. Os resultados são então passados para as páginas JSP para apresentação.

- Gerenciamento pelo Contêiner: O contêiner EJB gerencia a criação, pool e ciclo de vida dos EJBs, incluindo aspectos como transações e segurança, permitindo que os desenvolvedores se concentrem na lógica de aplicação.

Segue abaixo os códigos do arquivo ProdutoLista.jsp:

```

1  <!--
2      Document : ProdutoListia
3      Created on : May 26, 2024, 7:47:57 PM
4      Author : Alaila
5      -->
6  <@page import="cadastros.modelo.Produto">
7  <@page request="java.util.List">
8  <@ page contentType="text/html; charset=UTF-8" language="java" %>
9  <DOCTYPE html>
10  <html>
11  <head>
12  <title>Lista de Produtos</title>
13  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet">
14  </head>
15  <body>
16  <div class="container">
17  <div class="mt-4">
18  <a href="/ServletProdutoFacaoFormIncluir" class="btn btn-primary mb-3">Incluir Novo Produto</a>
19
20  <table class="table table-bordered">
21  <thead class="table-dark">
22  <tr>
23  <th>Id</th>
24  <th>Nome</th>
25  <th>Quantidade</th>
26  <th>Preço</th>
27  <th>Acões</th>
28  </tr>
29  </thead>
30  <tbody>
31  <
32  <tr>
33  <td colspan="5">
34  <div>
35  <div>
36  <div>
37  <div>
38  <div>
39  <div>
40  <div>
41  <div>
42  <div>
43  <div>
44  <div>
45  <div>
46  <div>
47  <div>
48  <div>
49  <div>
50  <div>
51  <div>
52  <div>
53  <div>
54  <div>
55  <div>
56  <div>
57  <div>
58  <div>
59  <div>
60  <div>
61  <div>
62  <div>
63  <div>
64  <div>
65  <div>
66  <div>
67  <div>
68  <div>
69  <div>
70  <div>
71  <div>
72  <div>
73  <div>
74  <div>
75  <div>
76  <div>
77  <div>
78  <div>
79  <div>
80  <div>
81  <div>
82  <div>
83  <div>
84  <div>
85  <div>
86  <div>
87  <div>
88  <div>
89  <div>
90  <div>
91  <div>
92  <div>
93  <div>
94  <div>
95  <div>
96  <div>
97  <div>
98  <div>
99  <div>
100  <div>
101  <div>
102  <div>
103  <div>
104  <div>
105  <div>
106  <div>
107  <div>
108  <div>
109  <div>
110  <div>
111  <div>
112  <div>
113  <div>
114  <div>
115  <div>
116  <div>
117  <div>
118  <div>
119  <div>
120  <div>
121  <div>
122  <div>
123  <div>
124  <div>
125  <div>
126  <div>
127  <div>
128  <div>
129  <div>
130  <div>
131  <div>
132  <div>
133  <div>
134  <div>
135  <div>
136  <div>
137  <div>
138  <div>
139  <div>
140  <div>
141  <div>
142  <div>
143  <div>
144  <div>
145  <div>
146  <div>
147  <div>
148  <div>
149  <div>
150  <div>
151  <div>
152  <div>
153  <div>
154  <div>
155  <div>
156  <div>
157  <div>
158  <div>
159  <div>
160  <div>
161  <div>
162  <div>
163  <div>
164  <div>
165  <div>
166  <div>
167  <div>
168  <div>
169  <div>
170  <div>
171  <div>
172  <div>
173  <div>
174  <div>
175  <div>
176  <div>
177  <div>
178  <div>
179  <div>
180  <div>
181  <div>
182  <div>
183  <div>
184  <div>
185  <div>
186  <div>
187  <div>
188  <div>
189  <div>
190  <div>
191  <div>
192  <div>
193  <div>
194  <div>
195  <div>
196  <div>
197  <div>
198  <div>
199  <div>
200  <div>
201  <div>
202  <div>
203  <div>
204  <div>
205  <div>
206  <div>
207  <div>
208  <div>
209  <div>
210  <div>
211  <div>
212  <div>
213  <div>
214  <div>
215  <div>
216  <div>
217  <div>
218  <div>
219  <div>
220  <div>
221  <div>
222  <div>
223  <div>
224  <div>
225  <div>
226  <div>
227  <div>
228  <div>
229  <div>
230  <div>
231  <div>
232  <div>
233  <div>
234  <div>
235  <div>
236  <div>
237  <div>
238  <div>
239  <div>
240  <div>
241  <div>
242  <div>
243  <div>
244  <div>
245  <div>
246  <div>
247  <div>
248  <div>
249  <div>
250  <div>
251  <div>
252  <div>
253  <div>
254  <div>
255  <div>
256  <div>
257  <div>
258  <div>
259  <div>
260  <div>
261  <div>
262  <div>
263  <div>
264  <div>
265  <div>
266  <div>
267  <div>
268  <div>
269  <div>
270  <div>
271  <div>
272  <div>
273  <div>
274  <div>
275  <div>
276  <div>
277  <div>
278  <div>
279  <div>
280  <div>
281  <div>
282  <div>
283  <div>
284  <div>
285  <div>
286  <div>
287  <div>
288  <div>
289  <div>
290  <div>
291  <div>
292  <div>
293  <div>
294  <div>
295  <div>
296  <div>
297  <div>
298  <div>
299  <div>
300  <div>
301  <div>
302  <div>
303  <div>
304  <div>
305  <div>
306  <div>
307  <div>
308  <div>
309  <div>
310  <div>
311  <div>
312  <div>
313  <div>
314  <div>
315  <div>
316  <div>
317  <div>
318  <div>
319  <div>
320  <div>
321  <div>
322  <div>
323  <div>
324  <div>
325  <div>
326  <div>
327  <div>
328  <div>
329  <div>
330  <div>
331  <div>
332  <div>
333  <div>
334  <div>
335  <div>
336  <div>
337  <div>
338  <div>
339  <div>
340  <div>
341  <div>
342  <div>
343  <div>
344  <div>
345  <div>
346  <div>
347  <div>
348  <div>
349  <div>
350  <div>
351  <div>
352  <div>
353  <div>
354  <div>
355  <div>
356  <div>
357  <div>
358  <div>
359  <div>
360  <div>
361  <div>
362  <div>
363  <div>
364  <div>
365  <div>
366  <div>
367  <div>
368  <div>
3
```

Imagem 19 – ProdutoLista.jsp

[illegible]

Imagem 20 – ProdutoLista.jsp

Análise e Conclusão

Funcionamento do Padrão Front Controller na Arquitetura MVC

Padrão Front Controller:

- **Definição:** O padrão Front Controller é um modelo de design utilizado em aplicações web para gerir todas as requisições através de um ponto central de entrada. Este controlador centraliza a lógica de tratamento das requisições e as direciona aos componentes apropriados para processamento.
- **Arquitetura MVC:** Na arquitetura MVC (Model-View-Controller), o Front Controller serve como o controlador principal que recebe todas as requisições do cliente, determina a ação necessária, interage com os modelos (Model) para manipulação de dados e seleciona a visão (View) adequada para renderizar a resposta.

Implementação em Aplicações Web Java:

- Servlet Front Controller: Um servlet, como o ServletProdutoFC, é frequentemente usado como Front Controller. Ele captura todas as requisições e utiliza parâmetros para determinar a ação a ser executada.
- Diferenças e Semelhanças entre Servlets e JSPs

Servlets:

- Definição: Servlets são classes Java que expandem a funcionalidade de um servidor web, processando requisições e gerando respostas dinamicamente.
- Lógica de Negócio: São usados principalmente para controlar a lógica de negócios e manipulação de requisições HTTP.
- Desenvolvimento: Requerem escrita de código Java, são mais verbosos e menos intuitivos para a criação de interfaces de usuário.

JSPs (JavaServer Pages):

- Definição: JSPs são páginas HTML que podem conter trechos de código Java, permitindo a criação de conteúdo dinâmico.
- Interação com o Usuário: São mais adequados para a apresentação e renderização de interfaces de usuário.
- Desenvolvimento: Permitem a incorporação de tags JSP e JSTL (JavaServer Pages Standard Tag Library), facilitando a criação de páginas web dinâmicas com menos código Java.

Semelhanças:

- Ambos são componentes server-side usados para criar aplicações web dinâmicas.
- Ambos utilizam o protocolo HTTP para comunicação com clientes (navegadores).
- Diferença entre Redirecionamento Simples e Método Forward

Redirecionamento Simples (sendRedirect):

- Comportamento: Envia uma resposta HTTP ao cliente, instruindo-o a fazer uma nova requisição para uma URL diferente.
- Exemplo de Uso: Quando é necessário redirecionar o usuário para uma página diferente após o processamento de uma ação (por exemplo, redirecionar para uma página de login após logout).

- Ciclo de Requisição: Inicia uma nova requisição HTTP, o que significa que o cliente verá a nova URL no navegador e qualquer dado da requisição anterior será perdido.

Método Forward (RequestDispatcher):

- Comportamento: Encaminha a requisição atual do servidor para outro recurso (servlet, JSP, etc.) sem que o cliente saiba.
- Exemplo de Uso: Quando é necessário passar o controle de uma requisição para outro componente sem alterar a URL no navegador do cliente.
- Ciclo de Requisição: Mantém a mesma requisição e resposta, permitindo que dados de requisição (parâmetros e atributos) sejam compartilhados.

Parâmetros e Atributos nos Objetos HttpRequest

Parâmetros:

- Definição: São valores enviados pelo cliente (navegador) como parte de uma requisição HTTP. Podem ser enviados via URL (GET) ou no corpo da requisição (POST).
- Uso: Utilizados para capturar dados enviados pelo cliente, como dados de formulários.

Atributos:

- Definição: São objetos armazenados no escopo da requisição e podem ser usados para passar dados entre diferentes componentes do lado do servidor durante o processamento de uma requisição.
- Uso: Utilizados para compartilhar dados entre servlets, JSPs e outros componentes web.

Com base em tudo o que foi feito anterior na parte de configurar toda a estrutura dos arquivos, atributos, pacotes, foi implementado e estilizado as páginas, com um layout mais moderno, com melhor apresentação e responsivo. Foi utilizado o framework Bootstrap nos arquivos ProdutoLista.jsp e ProdutoDados.jsp, ficando dessa seguinte forma:

localhost:8080/CadastroE-ww/SendProduto/Cacao

Lista de Produtos

[Incluir Novo Produto](#)

#	Nome	Quantidade	Preço	Ações
3	Manga	800	4.0	Alterar Excluir
4	Pêra	450	9.0	Alterar Excluir
5	Uva	400	5.6	Alterar Excluir
6	Morango	350	8.0	Alterar Excluir
7	Carambola	600	3.5	Alterar Excluir
14	Banana	100	5.0	Alterar Excluir
15	Laranja	500	2.0	Alterar Excluir
19	Castanha	365	10.0	Alterar Excluir
21	maçã	785	5.0	Alterar Excluir

Imagem 21 – Página de lista dos produtos

Cadastro de Produto

[Voltar](#)

Nome

Quantidade

Preço de Venda

[Cadastrar](#)

Imagem 22 – Página de Cadastro de Produtos

Como o Framework Bootstrap é Utilizado?

Framework Bootstrap:

- É um framework de código aberto para o desenvolvimento de interfaces web responsivas e adaptadas para dispositivos móveis. Ele oferece um conjunto de ferramentas para criar páginas web e aplicações com um design uniforme.

Porque utilizar o bootstrap:

- Inclusão via CDN: Bootstrap pode ser facilmente integrado em um projeto web ao incluir os links CSS e JavaScript via Content Delivery Network (CDN) nas páginas HTML ou JSP.
- Componentes Pré-Estilizados: Bootstrap disponibiliza uma ampla variedade de componentes pré-estilizados, como botões, formulários,

tabelas, e menus de navegação, que podem ser incorporados e personalizados conforme necessário.

- **Classes Utilitárias:** Bootstrap inclui diversas classes utilitárias para espaçamento, alinhamento, exibição, facilitando o ajuste rápido do layout sem necessidade de CSS personalizado.

Por que o Bootstrap Garante a Independência Estrutural do HTML?

Independência Estrutural do HTML:

- **Classes Pré-Definidas:** Bootstrap utiliza um sistema de classes CSS pré-definidas que podem ser aplicadas diretamente aos elementos HTML. Isso permite que a estrutura do HTML permaneça simples, enquanto o estilo e o comportamento são controlados através das classes do Bootstrap.
- **Separação de Preocupações:** Utilizando classes do Bootstrap, a preocupação com o estilo é separada da estrutura HTML. Isso facilita a manutenção e a atualização do código, pois as mudanças de estilo podem ser feitas no CSS sem modificar o HTML.
- **Consistência e Reutilização:** Bootstrap promove a consistência no design através do uso de suas classes padrão, permitindo que desenvolvedores reutilizem componentes uniformemente em toda a aplicação sem precisar redefinir estilos repetidamente.

Relação entre o Bootstrap e a Responsividade da Página

Responsividade com Bootstrap:

- **Design Responsivo:** Bootstrap foi projetado com uma abordagem mobile-first, priorizando a experiência do usuário em dispositivos móveis. A responsividade é um dos pilares do Bootstrap, garantindo que as páginas web se adaptem a diferentes tamanhos de tela e dispositivos.
- **Sistema de Grid Flexível:** O sistema de grid do Bootstrap permite a criação de layouts flexíveis e responsivos. Utilizando classes de colunas e linhas, os desenvolvedores podem definir como o conteúdo deve se comportar em várias larguras de tela.
- **Media Queries e Breakpoints:** Bootstrap utiliza media queries para aplicar diferentes estilos em diferentes tamanhos de tela. Os breakpoints predefinidos (como sm, md, lg, xl) ajudam a definir comportamentos específicos para vários dispositivos, desde smartphones até desktops.
- **Componentes Responsivos:** Muitos componentes do Bootstrap, como menus de navegação, modais e carrosséis, são projetados para funcionar de forma responsiva automaticamente, proporcionando uma experiência de usuário coesa em todas as plataformas.