MISSÃO PRÁTICA – NÍVEL 4

Aluno: Daniel Kilzer Brasil Dias | **Matrícula:** 202310422026

Campus: Polo Bezerra de Menezes – Fortaleza/CE
Disciplina: Vamos Integrar Sistemas **– Turma:** 9001 **– Semestre:** 2024.4

Objetivo da Prática

O objetivo desta prática foi desenvolver um sistema cadastral web robusto, utilizando tecnologias *Java Enterprise Edition* (JEE) para consolidar conceitos essenciais de desenvolvimento de software corporativo. A aplicação, focada no gerenciamento de produtos, foi construída com base em uma arquitetura MVC (*Model-View-Controller*), integrando Servlets, JPA (*Java Persistance* API), EJB (*Enterprise JavBeans*) e JSP (*Java Server Page*) combinado com *Bootstrap*.

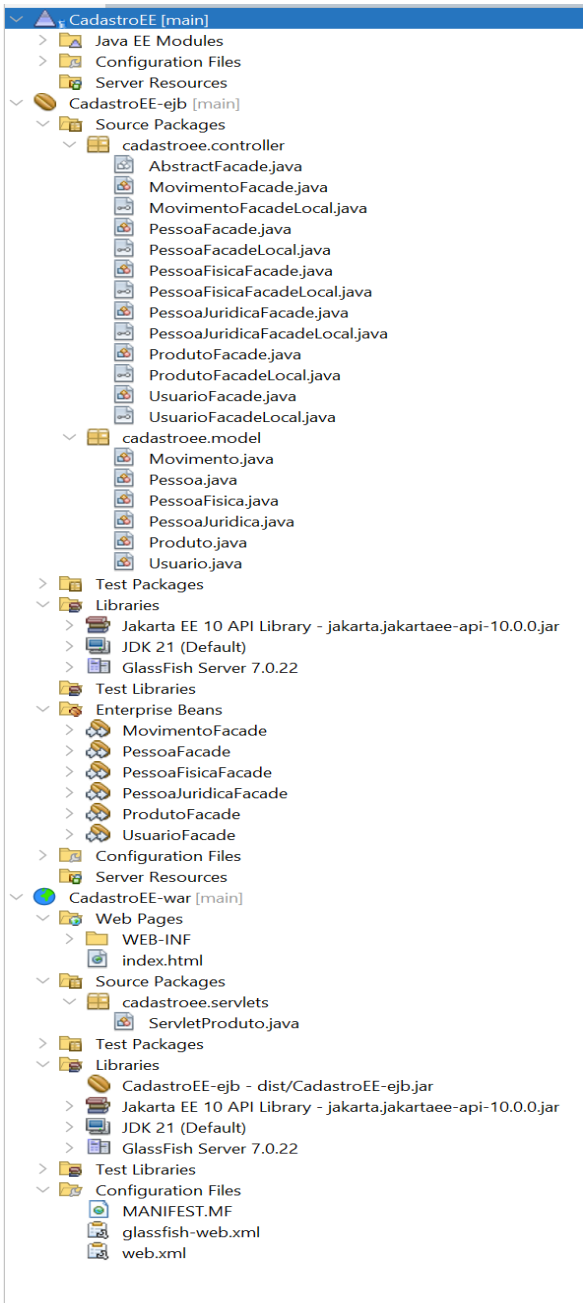
A prática visou demonstrar a importância de padrões de projeto como *Front Controller* e *Data Access Object* (DAO), além de ressaltar a modularização proporcionada pela plataforma JEE. Através da integração entre Servlets, JPA e EJBs, o sistema implementa operações CRUD (*Create, Read, Update, Delete*), garantindo escalabilidade e separação clara entre camadas de apresentação, controle e dados.

1º Procedimento | Camadas de Persistência e Controle

CÓDIGOS

Árvore de diretórios

A organização dos diretórios do projeto CadastroEE, depois de concluídas as etapas do 1º Procedimento, é a da figura que segue.



A seguir, são apresentados os principais códigos solicitados nesse procedimento. Destaque-se que alguns desses códigos foram gerados automaticamente com o auxílio das ferramentas disponibilizadas pela IDE NetBeans 24.

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
             xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence

https://jakarta.ee/xml/ns/persistence/persistence_3_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>
            <property name="jakarta.persistence.schema-
generation.database.action" value="update"/>
        </properties>
    </persistence-unit>

</persistence>
```

web.xml

```
<?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>ServletProduto</servlet-name>
        <servlet-
class>cadastroee.servlets.ServletProduto</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>ServletProdutoFC</servlet-name>
        <servlet-
class>cadastroee.servlets.ServletProdutoFC</servlet-class>
    </servlet>
```

```
    <!-- Mapeamento de URLs -->
    <servlet-mapping>
        <servlet-name>ServletProduto</servlet-name>
        <url-pattern>/ServletProduto</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ServletProdutoFC</servlet-name>
        <url-pattern>/ServletProdutoFC</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

Pacote `cadastroee.model` | Classe `Pessoa`

```
package cadastroee.model;

import jakarta.persistence.Basic;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
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 java.io.Serializable;

@Entity
@Table(name = "PessoaFisica")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "PessoaFisica.findAll", query =
"SELECT p FROM PessoaFisica p"),
    @NamedQuery(name = "PessoaFisica.findByIdPessoa", query
= "SELECT p FROM PessoaFisica p WHERE p.idPessoa
= :idPessoa"),
    @NamedQuery(name = "PessoaFisica.findByCpf", query =
"SELECT p FROM PessoaFisica p WHERE p.cpf = :cpf")})
public class PessoaFisica implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "idPessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 11)
    @Column(name = "cpf")
    private String cpf;
    @JoinColumn(name = "idPessoa", referencedColumnName =
"idPessoa", insertable = false, updatable = false)
    @OneToOne(optional = false)
    private Pessoa pessoa;

    public PessoaFisica() {
    }

    public PessoaFisica(Integer idPessoa) {
        this.idPessoa = idPessoa;
    }

    public PessoaFisica(Integer idPessoa, String cpf) {
        this.idPessoa = idPessoa;
        this.cpf = cpf;
    }

    public Integer getIdPessoa() {
        return idPessoa;
    }

    public void setIdPessoa(Integer idPessoa) {
        this.idPessoa = idPessoa;
    }

    public String getCpf() {
        return cpf;
    }

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

    public Pessoa getPessoa() {
        return pessoa;
    }
```

```
    public void setPessoa(Pessoa pessoa) {
        this.pessoa = pessoa;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idPessoa != null ? idPessoa.hashCode() :
0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof PessoaFisica)) {
            return false;
        }
        PessoaFisica other = (PessoaFisica) object;
        if ((this.idPessoa == null && other.idPessoa !=
null) || (this.idPessoa != null &&
this.idPessoa.equals(other.idPessoa))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "cadastroee.model.PessoaFisica[ idPessoa=" +
idPessoa + " ]";
    }

}
```

Pacote `cadastroee.servlets` | Servlet `ServletProduto`

O código de ServletProduto foi gerado automaticamente, mas também modificado em partes para atender aos requisitos do projeto.

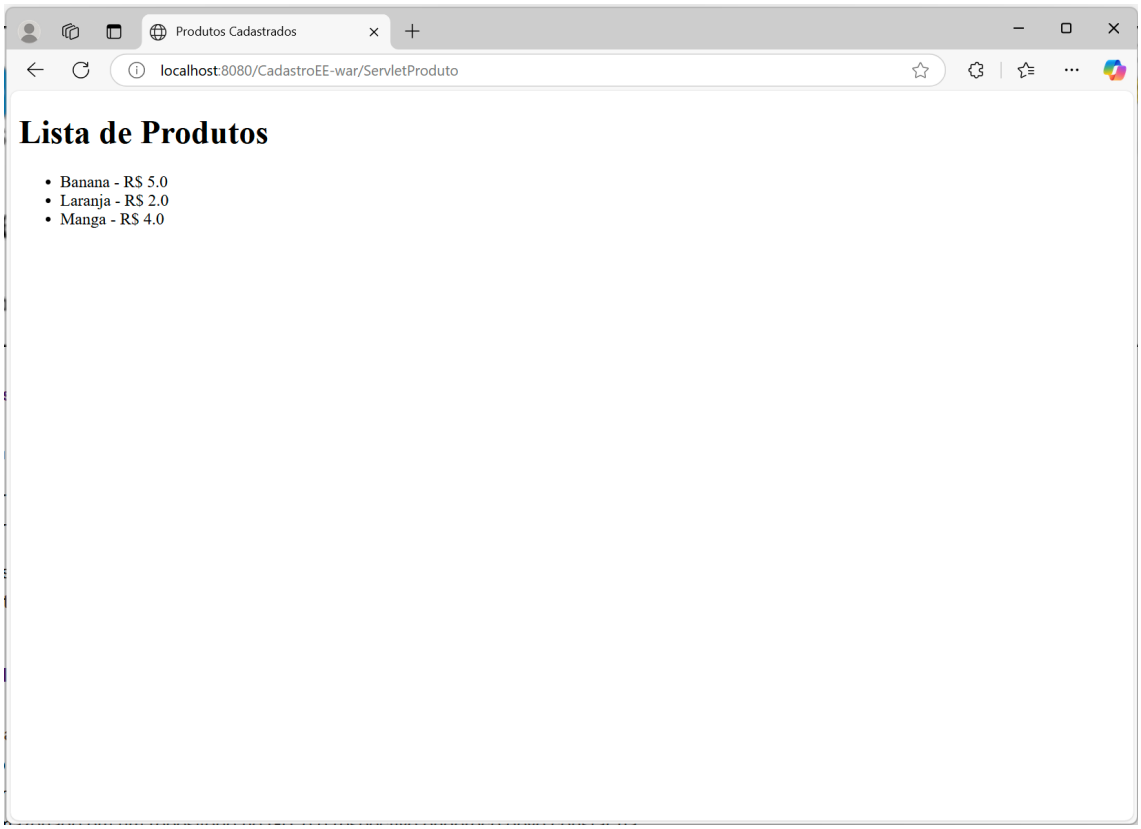
```
package cadastroee.servlets;

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;

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

            // Recupera a lista de produtos via EJB.
            List<Produto> produtos = facade.findAll();

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

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

    @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 "Short description";
    }
}
```

RESULTADOS

Após a execução do projeto, cujos códigos expostos até aqui são apenas amostras, temos o resultado apresentado na figura a seguir, que mostra uma página Web com a lista de produtos cadastrados no banco de dados.



ANÁLISE

a) **Como é organizado um projeto corporativo no NetBeans?**

O NetBeans organiza projetos corporativos em uma estrutura modular. Um projeto típico é dividido em módulos distintos, cada um com responsabilidades específicas:

- Módulo EJB: Contém a lógica de negócios e os componentes Enterprise JavaBeans
- Módulo Web: Responsável pela interface com o usuário, contendo Servlets, JSPs e recursos web
- Módulo de Entidades: Armazena as classes de domínio e configurações JPA
- Módulo Enterprise Application: Agrupa todos os outros módulos em um único arquivo EAR para deploy

O NetBeans automatiza a criação desta estrutura através de wizards, gerando automaticamente os arquivos de configuração necessários (web.xml, persistence.xml, etc.) e mantendo as dependências entre os módulos.

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

- **JPA** (Java Persistence API): define um padrão para o mapeamento objeto-relacional (ORM), permitindo que os desenvolvedores persistam dados em bancos de dados relacionais usando objetos Java. Ela simplifica o acesso a dados, reduzindo a necessidade de escrever código SQL diretamente.
- **EJB** (Enterprise Java Beans): fornece uma arquitetura para o desenvolvimento de componentes de negócios robustos e escaláveis. Os EJBs podem ser Session Beans (stateless ou stateful) que implementam a lógica de negócios, ou Message-Driven Beans que consomem mensagens assíncronas. Eles

oferecem serviços como gerenciamento de transações, segurança e concorrência.

Em um aplicativo Web, os Servlets atuam como controladores, recebendo requisições do usuário e interagindo com os EJBs para processar a lógica de negócios. Os EJBs, por sua vez, utilizam JPA para acessar e manipular os dados no banco de dados.

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

O NetBeans oferece uma série de ferramentas e funcionalidades que agilizam o desenvolvimento com essas tecnologias, como:

- **Wizards e Templates:** auxiliam na criação de entidades JPA e componentes EJB, gerando o código *boilerplate* automaticamente.
- **Integração com Servidores de Aplicação:** permite *deploy*, testes e *debugging* integrados com servidores como GlassFish ou Tomcat.
- **Edição Facilitada de Arquivos de Configuração:** ferramentas visuais para editar arquivos como persistence.xml, evitando erros de configuração.
- ***Code Completion* e *Refactoring*:** recursos que ajudam a escrever e manter o código de forma mais eficiente, reduzindo a chance de erros e aumentando a produtividade.

Essas facilidades permitem que o desenvolvedor foque na implementação da lógica de negócio, economizando tempo e esforço na configuração e na integração das tecnologias.

d) 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 processam requisições HTTP de forma dinâmica. Eles atuam como controladores na arquitetura MVC web, recebendo requisições do cliente, processando-as e gerando respostas.

O NetBeans oferece suporte para Servlets através de:

- Disponibilização de templates e assistentes que simplificam a criação de novos Servlets.
- Automatização da configuração do web.xml ou utilização de anotações para mapear os Servlets.
- Integração com servidores de aplicação para testes e deploy, além de ferramentas de debugging que facilitam o desenvolvimento.

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

A comunicação entre os Servlets e os Session Beans, que fazem parte do pool de EJBs, pode ser realizada basicamente de duas maneiras:

- **Injeção via @EJB:**
  - Nos Servlets, é possível utilizar a anotação @EJB para injetar automaticamente a referência ao Session Bean.
  - Essa abordagem elimina a necessidade de lookup manual, facilitando o acesso aos métodos de negócio.
- **Lookup via JNDI:** em situações onde a injeção não é aplicável ou desejada, pode-se realizar uma busca (*lookup*) através do JNDI (*Java Naming and Directory Interface*) para obter a referência do EJB.

Em ambos os casos, o Servlet atua como o cliente, chamando os métodos do EJB para executar a lógica de negócio, enquanto o container gerencia o ciclo de vida, transações e outras funcionalidades essenciais dos EJBs.

2º Procedimento | Interface Cadastral com Servlet e JSPs

CÓDIGOS

A seguir, são apresentados os principais códigos solicitados nesse procedimento.

ServletProdutoFC

```
package cadastroee.servlets;

import cadastroee.controller.ProdutoFacadeLocal;
import cadastroee.model.Produto;
import jakarta.ejb.EJB;
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 java.io.IOException;

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

    @EJB
    private ProdutoFacadeLocal facade;

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

        String acao = request.getParameter("acao");
        if (acao == null) acao = "listar";

        String destino;

        switch (acao) {
            case "formIncluir":
            case "formAlterar":
                destino = "ProdutoDados.jsp";
                break;
            default:
                destino = "ProdutoLista.jsp";
```

```
        }

        try {
            switch (acao) {
                case "listar":
                    request.setAttribute("listaProdutos",
facade.findAll());
                    break;

                case "formAlterar":
                    Integer idAlterar =
Integer.parseInt(request.getParameter("id"));
                    Produto produtoAlterar =
facade.find(idAlterar);
                    request.setAttribute("produto",
produtoAlterar);
                    break;

                case "excluir":
                    Integer idExcluir =
Integer.parseInt(request.getParameter("id"));
                    Produto produtoExcluir =
facade.find(idExcluir);
                    facade.remove(produtoExcluir);
                    request.setAttribute("listaProdutos",
facade.findAll());
                    break;

                case "alterar":
                    Integer id =
Integer.parseInt(request.getParameter("id"));
                    Produto produto = facade.find(id);

produto.setNome(request.getParameter("nome"));

produto.setPrecoVenda(Float.parseFloat(request.getParameter
("precoVenda")));
                    facade.edit(produto);
                    request.setAttribute("listaProdutos",
facade.findAll());
                    break;

                case "incluir":
                    Produto novoProduto = new Produto();

novoProduto.setNome(request.getParameter("nome"));

novoProduto.setPrecoVenda(Float.parseFloat(request.getParam
eter("precoVenda")));
```

```
                    facade.create(novoProduto);
                    request.setAttribute("listaProdutos",
facade.findAll());
                    break;
                }
            } catch (NumberFormatException e) {
                throw new ServletException("Erro na conversão
de dados", e);
            }


request.getRequestDispatcher(destino).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);
    }
}
```

ProdutoDados

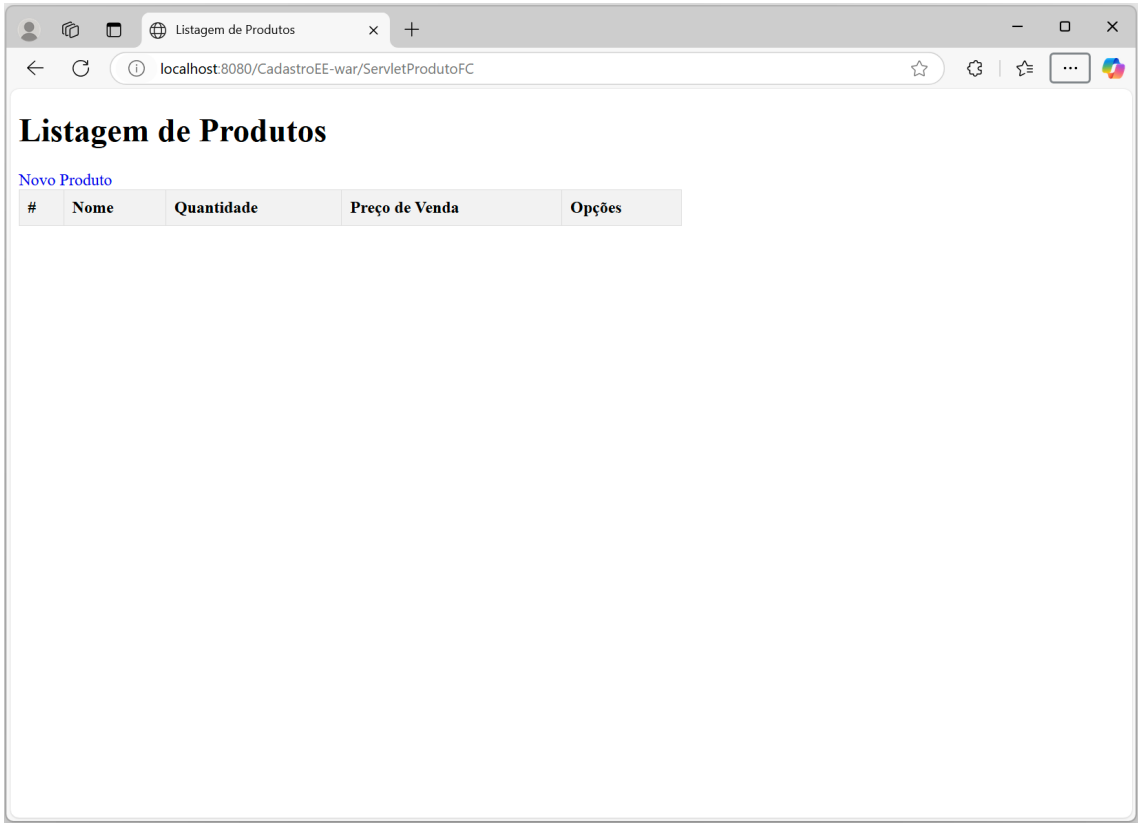
```
<%@ page language="java" contentType="text/html;
charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="model.Produto" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Dados do Produto</title>
    <style>
        .form-group {
            margin: 10px 0;
        }
        label {
            display: inline-block;
            width: 100px;
```

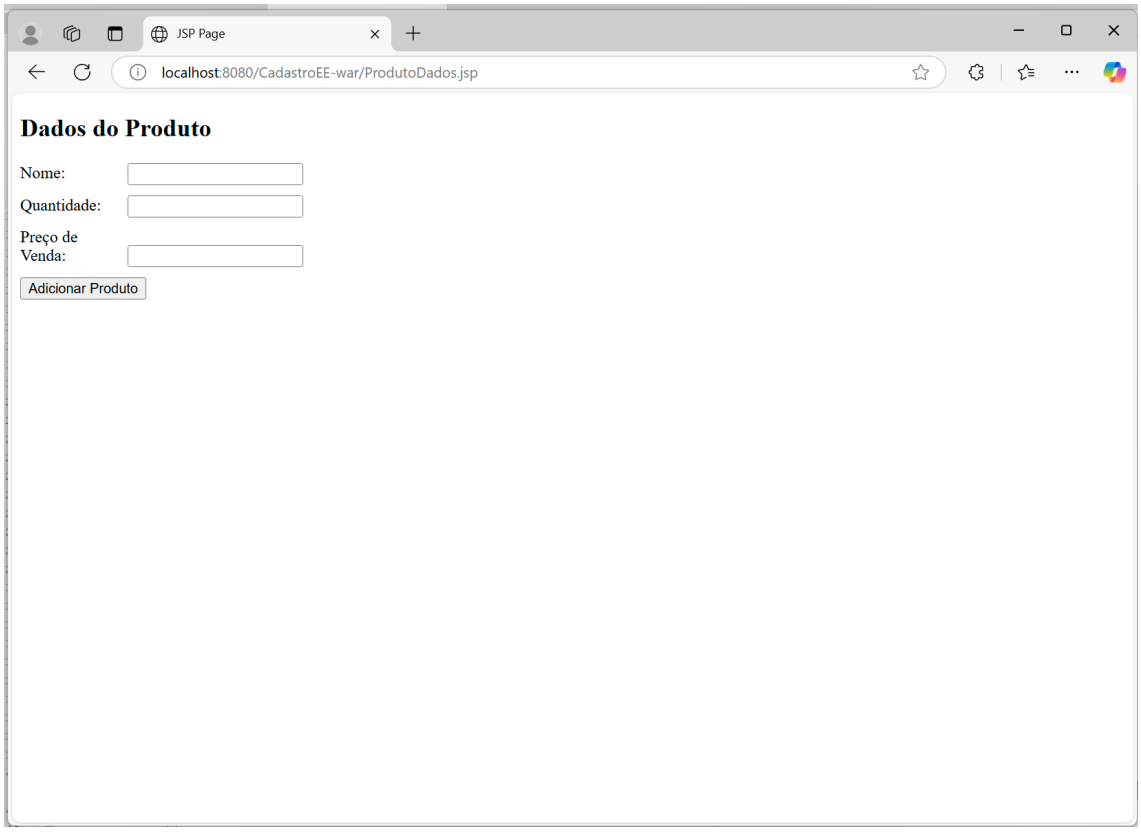
```
        }
    </style>
</head>
<body>
    <%
        Produto produto = (Produto)
request.getAttribute("produto");
        String acao = (produto == null) ? "incluir" :
"alterar";
    %>

    <h2>Dados do Produto</h2>

    <form action="ServletProdutoFC" method="post">
        <input type="hidden" name="acao" value="<%= acao
%>">

        <% if(acao.equals("alterar")) { %>
            <input type="hidden" name="id" value="<%=
produto.getId() %>">
        <% } %>

        <div class="form-group">
            <label for="nome">Nome:</label>
            <input type="text" id="nome" name="nome"
                   value="<%= (produto != null) ?
produto.getNome() : "" %>">
        </div>

        <div class="form-group">
            <label for="quantidade">Quantidade:</label>
            <input type="number" id="quantidade"
name="quantidade"
                   value="<%= (produto != null) ?
produto.getQuantidade() : "" %>">
        </div>

        <div class="form-group">
            <label for="precoVenda">Preço de Venda:</label>
            <input type="number" step="0.01"
id="precoVenda" name="precoVenda"
                   value="<%= (produto != null) ?
produto.getPrecoVenda() : "" %>">
        </div>

        <div class="form-group">
            <input type="submit" value="<%=
acao.equals("incluir") ? "Adicionar Produto" : "Alterar
Produto" %>">
```

```
        </div>
    </form>
</body>
</html>
```

RESULTADOS

Após a execução do projeto, temos o resultado apresentado na figura a seguir, que mostra uma página Web com a lista de produtos cadastrados no banco de dados, além da possibilidade de manipular os dados, incluindo, excluindo e alterando produto.





ANÁLISE

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

O *Front Controller* é um padrão de design que centraliza o tratamento de todas as requisições que chegam à aplicação. Em vez de cada recurso ter seu próprio ponto de entrada, um único componente – geralmente um `Servlet` – intercepta todas as requisições. Esse componente tem as seguintes responsabilidades:

- **Centralização do Controle:** Todas as requisições são direcionadas a esse único ponto, permitindo aplicar lógicas comuns (como autenticação, logging, e tratamento de exceções) de forma centralizada.
- **Descentralização da Lógica de Negócio:** Após analisar a requisição, o *Front Controller* decide qual ação (ou qual parte do

sistema) deve tratar a requisição, delegando a tarefa para outros componentes (por exemplo, classes de ação ou controladores específicos).

- **Integração com MVC:** Na arquitetura MVC, o *Front Controller* atua como o "Controlador" principal, coordenando a interação entre a camada de apresentação (*views*) e a camada de negócio (*model*). Em *frameworks* Java, esse papel é frequentemente desempenhado por componentes como o `DispatcherServlet` no Spring MVC ou o controlador central do Struts.

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

Servlets são componentes Java que executam no servidor Web e processam requisições HTTP. Eles são responsáveis por receber requisições, executar a lógica de negócios e gerar a resposta. Geralmente são usados para implementar a lógica de controle da aplicação.

As JSPs (*Java Server Pages*) são páginas HTML que contêm trechos de código Java. Elas são convertidas em Servlets pelo servidor web antes de serem executadas. São usadas para criar a interface de usuário da aplicação.

Semelhanças:

- Ambos são executados no servidor web e geram conteúdo dinâmico.
- Ambos podem acessar os recursos da plataforma Java.

Diferenças:

- Foco: Servlets focam na lógica de controle, enquanto JSPs focam na apresentação.
- Sintaxe: Servlets são escritos em Java, enquanto JSPs são páginas HTML com trechos de código Java.

- Ciclo de vida: Servlets têm um ciclo de vida gerenciado pelo servidor web, enquanto JSPs são convertidas em Servlets e seguem o mesmo ciclo de vida.

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?

Redirecionamento vs. Forward:

- Redirecionamento Simples (sendRedirect):
  - Funcionamento: o servidor envia uma resposta ao cliente (navegador) com o código de status HTTP 302, instruindo-o a fazer uma nova requisição para a URL especificada.
  - Consequências:
    - O endereço exibido no navegador é alterado para a nova URL.
    - Toda a requisição anterior é descartada; novos dados (parâmetros ou atributos) não são automaticamente transferidos.
    - É útil quando se deseja que o cliente saiba da mudança de endereço ou quando se precisa redirecionar para um recurso externo.
- Forward (RequestDispatcher.forward):
  - Funcionamento: o *container* encaminha a mesma requisição para outro recurso interno (como um Servlet ou uma JSP) sem que o cliente perceba essa mudança.
  - Consequências:
    - O URL no navegador permanece inalterado.

- Os parâmetros e atributos da requisição original são mantidos, permitindo a passagem de dados entre os componentes que participam do mesmo ciclo de requisição.
- É indicado para encaminhar a requisição para uma *view* ou para outro componente que fará o processamento final.

Parâmetros e Atributos em `HttpRequest`:

- Parâmetros (`request.getParameter`):
  - Vêm da requisição HTTP (URL ou POST)
  - São sempre strings
  - Somente leitura
  - Persistem apenas durante a requisição atual
  - Úteis para dados de formulários
- Atributos (`request.getAttribute`):
  - São definidos programaticamente.
  - Podem ser de qualquer tipo.
  - Podem ser modificados.
  - Disponíveis durante o ciclo de vida do `request`.
  - Úteis para passar dados entre componentes.

3º Procedimento | Melhorando o Design da Interface

CÓDIGOS

A seguir, os códigos das páginas ProdutoDados e ProdutoLista, após alterações de estilos via Bootstrap.

ProdutoDados.jsp

```
<%@ page language="java" contentType="text/html;
charset=UTF-8" pageEncoding="UTF-8"%>
<%@page import="cadastroee.model.Produto" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Dados do Produto</title>

        <!--Inclusão de Bootstrap-->
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css
/bootstrap.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+A
LEwIH" crossorigin="anonymous">
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/b
ootstrap.bundle.min.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6
jIeHz" crossorigin="anonymous"></script>
    </head>
    <body class="container">
        <%
            Produto produto = (Produto)
request.getAttribute("produto");
            String acao = (produto == null) ? "incluir" :
"alterar";
        %>

        <h2>Dados do Produto</h2>

        <form action="ServletProdutoFC" method="post"
class="form">
            <input type="hidden" name="acao" value="<%=
acao%>">

            <% if (acao.equals("alterar")) {%>
            <input type="hidden" name="id" value="<%=
produto.getIdProduto()%>">
            <% }%>

            <div class="mb-3">
                <label for="nome" class="form-
label">Nome:</label>
                <input type="text" id="nome" name="nome"
                       value="<%= (produto != null) ?
produto.getNome() : ""%>"
```

```
                        class="form-control">
            </div>

            <div class="mb-3">
                <label for="quantidade" class="form-
label">Quantidade:</label>
                <input type="number" id="quantidade"
name="quantidade"
                        value="<%= (produto != null) ?
produto.getQuantidade() : ""%>"
                        class="form-control">
            </div>

            <div class="mb-3">
                <label for="precoVenda" class="form-
label">Preço de Venda:</label>
                <input type="number" step="0.01"
id="precoVenda" name="precoVenda"
                        value="<%= (produto != null) ?
produto.getPrecoVenda() : ""%>"
                        class="form-control">
            </div>

            <div class="mb-3">
                <input type="submit"
                        value="<%= acao.equals("incluir") ?
"Adicionar Produto" : "Alterar Produto"%>"
                        class="btn btn-primary ">
            </div>
        </form>
    </body>
</html>
```

ProdutoLista.jsp

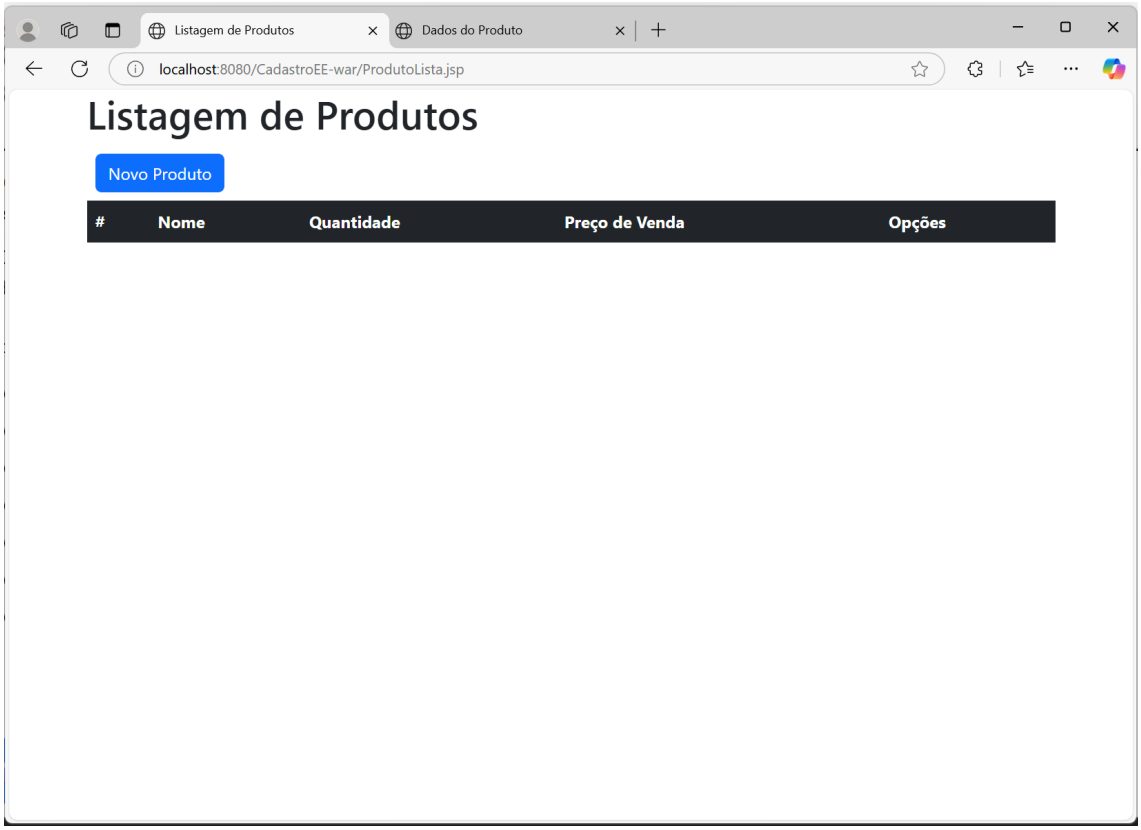
```
<%@page import="cadastroee.model.Produto"%>
<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Listagem de Produtos</title>

        <!--Inclusão de Bootstrap-->
```

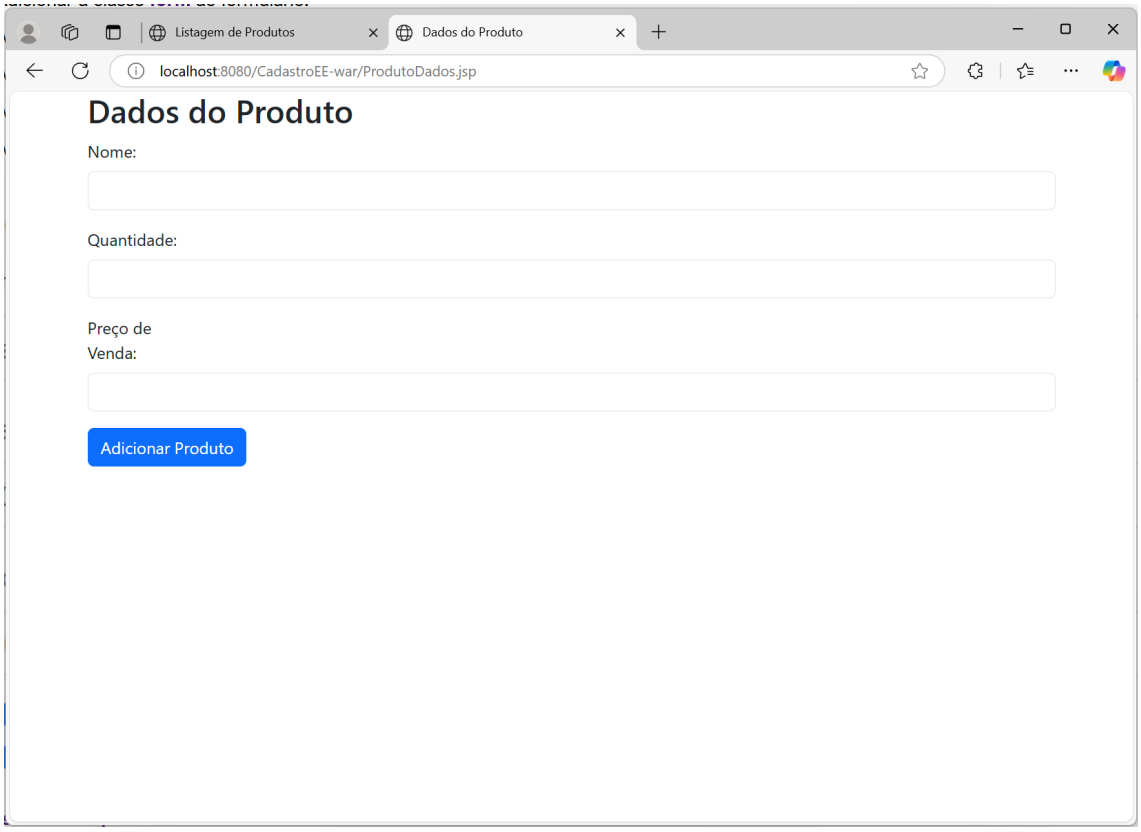
```
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css
/bootstrap.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+A
LEwIH" crossorigin="anonymous">
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/b
ootstrap.bundle.min.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6
jIeHz" crossorigin="anonymous"></script>
    </head>
    <body class="container">
        <h1>Listagem de Produtos</h1>

        <a href="produto?acao=formIncluir" class="btn btn-
primary m-2">Novo Produto</a>

        <table class="table table-striped">
            <thead class="table-dark">
                <tr>
                    <th>#</th>
                    <th>Nome</th>
                    <th>Quantidade</th>
                    <th>Preço de Venda</th>
                    <th>Opções</th>
                </tr>
            </thead>
            <tbody>
                <%
                    List<Produto> produtos =
(List<Produto>) request.getAttribute("produtos");

                    if (produtos != null) {
                        for (Produto produto : produtos) {
                %>

                <tr>
                    <td><%= produto.getIdProduto() %></td>
                    <td><%= produto.getNome() %></td>
                    <td><%= produto.getQuantidade() %></td>
                    <td><%= String.format("%.2f",
produto.getPrecoVenda()) %></td>
                    <td>
                        <a href="ServletProdutoFC?
acao=formAlterar&id=<%= produto.getIdProduto() %>"
class="btn btn-primary btn-sm">
                            Alterar
                        </a>
```

```
                            <a href="ServletProdutoFC?
acao=excluir&id=<%= produto.getIdProduto() %>" class="btn
btn-danger btn-sm">
                                Excluir
                            </a>
                        </td>
                    </tr>

                    <%
                            }
                        }
                    %>
                </tbody>
            </table>
        </body>
</html>
```

RESULTADOS

A seguir, imagens demonstrando o resultado da execução dos códigos.





ANÁLISE

a) Como o *framework Bootstrap* é utilizado?

O Bootstrap é um framework front-end amplamente utilizado para desenvolvimento web responsivo e ágil. Ele oferece uma coleção abrangente de componentes CSS e JavaScript pré-estilizados, que facilitam a criação de interfaces de usuário consistentes e personalizáveis.

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

O Bootstrap promove a independência estrutural do HTML, pois ele se concentra na estilização dos elementos, e não na sua estrutura. Isso significa que você pode usar o Bootstrap para estilizar qualquer tipo de estrutura HTML, sem precisar se preocupar em como os elementos estão organizados.

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

A responsividade é um dos principais pilares do *Bootstrap*. Ele foi projetado para facilitar a criação de páginas web que se adaptam a diferentes tamanhos de tela e dispositivos, como *desktops*, *tablets* e *smartphones*.

- Sistema de *grid*: utiliza um sistema de *grid* responsivo, baseado em linhas e colunas, que permite criar layouts flexíveis e adaptáveis. As colunas se ajustam automaticamente ao tamanho da tela, garantindo que o conteúdo seja exibido de forma adequada em qualquer dispositivo.
- Classes responsivas: oferece classes CSS que permitem controlar a exibição de elementos em diferentes tamanhos de tela.
- *Mobile-first*: adota a abordagem *mobile-first*, o que significa que os estilos são definidos primeiro para dispositivos móveis e, em seguida, adaptados para telas maiores. Isso garante que a experiência do usuário seja otimizada em dispositivos móveis, que são cada vez mais utilizados para acessar a web.

Conclusão

A implementação do sistema cadastral permitiu consolidar conhecimentos críticos no ecossistema Java EE. Utilizando Servlets como núcleo de controle, JPA para abstração do banco de dados e EJBs para gerenciamento transacional, o projeto demonstrou a eficácia de uma arquitetura MVC em promover organização e reusabilidade de código.

A interface, desenvolvida com JSPs e *Bootstrap*, garantiu responsividade e consistência visual, enquanto o padrão *Front Controller* centralizou o roteamento, simplificando a manutenção.

Como resultado, o sistema não apenas cumpre os requisitos funcionais de cadastro, mas também serve como modelo para aplicações futuras, ilustrando boas práticas como:

1. Separação de responsabilidades entre camadas;
2. Uso de injeção de dependências (@EJB) para acoplamento flexível;
3. Adoção de *frameworks front-end* (*Bootstrap*) para agilizar o desenvolvimento.

Esta experiência reforça o potencial do Java EE para sistemas complexos, combinando robustez e modularidade.