

Disciplina: RPG0017 – Vamos integrar sistemas

Nome: João Gilberto dos Santos

Turma: 2022.4

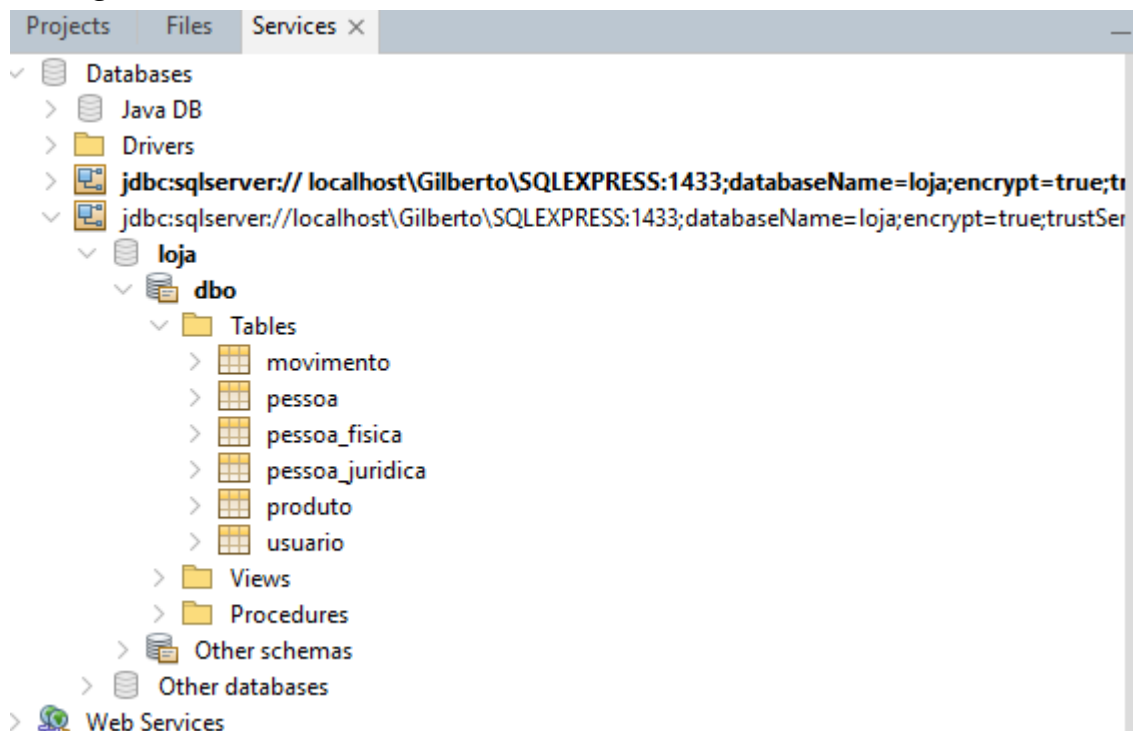
1º Título da Prática: Camadas de Persistência e Controle

2º Objetivo da Prática

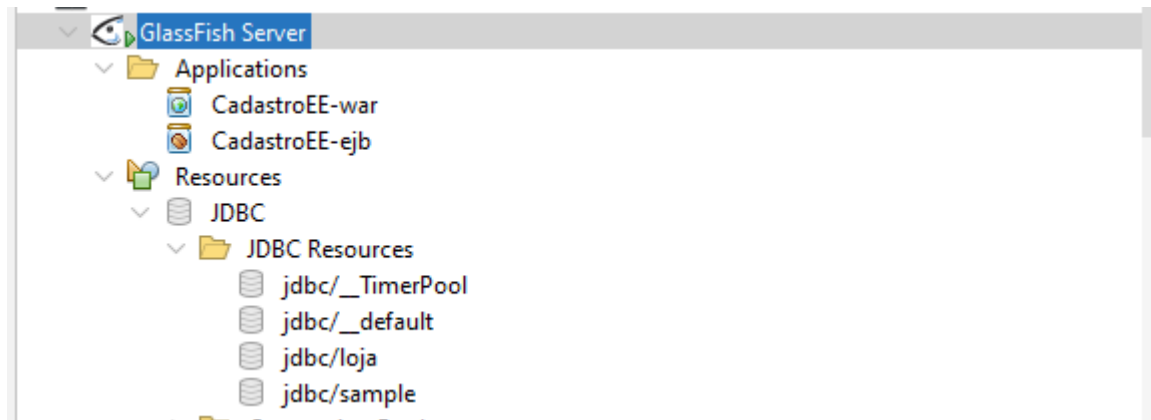
- Implementar persistência com base em JPA.
- Implementar regras de negócio na plataforma JEE, através de EJBs.
- Implementar sistema cadastral Web com base em Servlets e JSPs.
- Utilizar a biblioteca Bootstrap para melhoria do design.
- No final do exercício, o aluno terá criado todos os elementos necessários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

Procedimento / Camadas de Persistência e Controle

1.1 Configurar a conexão com SQL Server via NetBeans



1.2 Pool de conexões no GlassFish Server 6.2.1:



2.1 Aplicação do CadastroEE no Glasfisk

Home About...
User: admin Domain: domain1 Server: localhost
Eclipse GlassFish

Common Tasks

- Domain
 - server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
 - CadastroEE-ejb
 - CadastroEE-war**
- Lifecycle Modules
- Monitoring Data
- Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
- Configurations
 - default-config
 - server-config

General Descriptor

Edit Application

Modify an existing application or module.

Name: CadastroEE-war

Status: ☒

Virtual Servers:

server

Associates an Internet domain name with a physical server.

Context Root: /CadastroEE-war
Path relative to server's base URL. If empty, takes the default context path of a web application.

Implicit CDI ☒
Implicit discovery of CDI beans

Location: file:/C:/WorkSpace/CadastroEE/CadastroEE/CadastroEE-war/build/web/

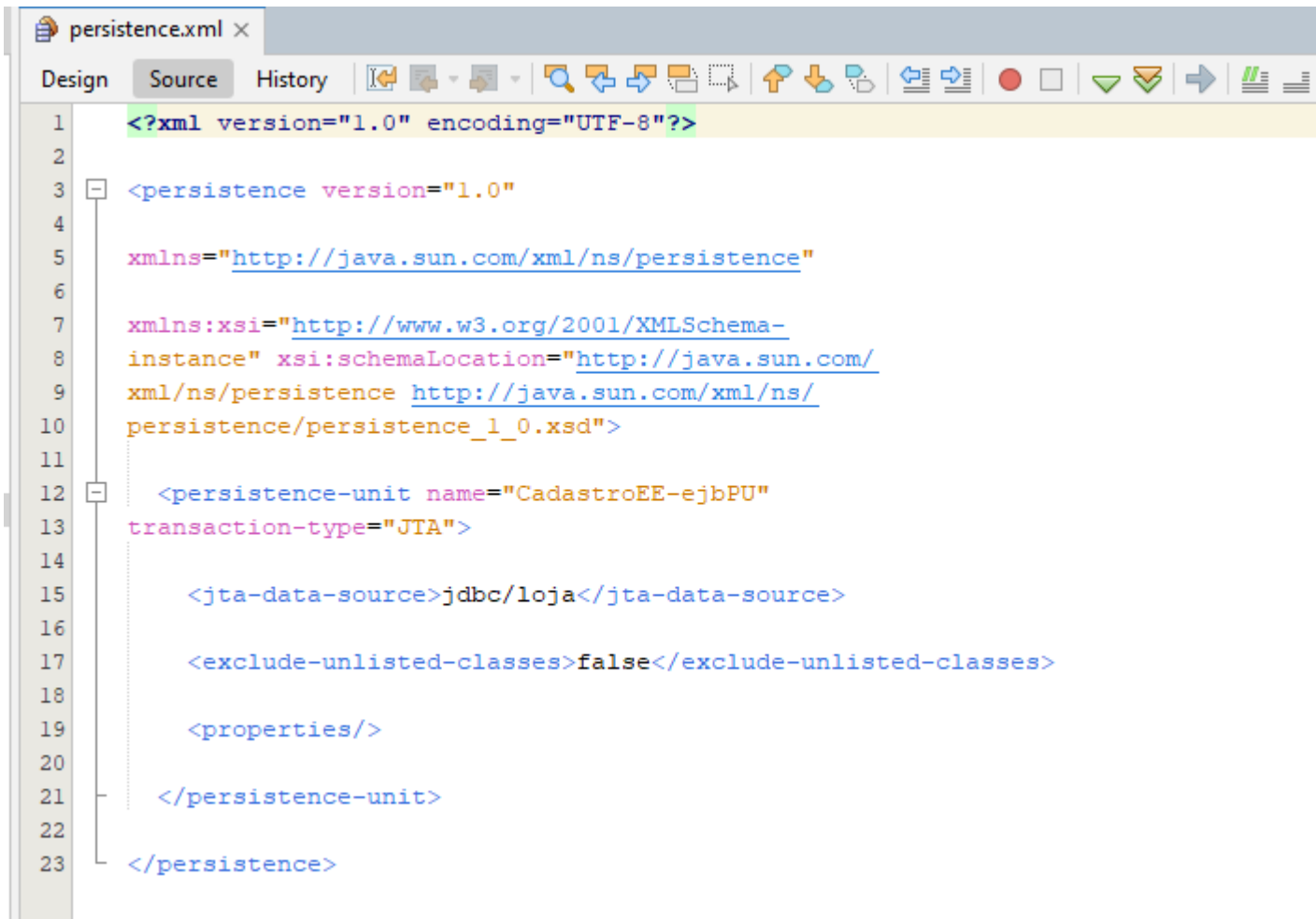
Deployment Order: 100
A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The default is 100

Libraries:

Description:

Modules and Components (4)		
Module Name	Engines	Component Name
CadastroEE-war	[web]	-----
CadastroEE-war		default
CadastroEE-war		jsp
CadastroEE-war		ServletProduto

3.1 arquivo de configuração do persistence.xml



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <persistence version="1.0"
4
5   xmlns="http://java.sun.com/xml/ns/persistence"
6
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
8   instance" xsi:schemaLocation="http://java.sun.com/
9   xml/ns/persistence http://java.sun.com/xml/ns/
10  persistence/persistence_1_0.xsd">
11
12   <persistence-unit name="CadastroEE-ejbPU"
13   transaction-type="JTA">
14
15     <jta-data-source>jdbc/loja</jta-data-source>
16
17     <exclude-unlisted-classes>>false</exclude-unlisted-classes>
18
19     <properties/>
20
21   </persistence-unit>
22
23 </persistence>
```

4.1 Servlet em produção

← ↻ 🏠 ⓘ localhost:8080/CadastroEE-war/ServletProduto	
<h1>Lista de Produtos</h1>	
Nome do Produto	Preço
Banana	R\$ 5,00
Laranja	R\$ 2,00
Manga	R\$ 4,00
Tangerina	R\$ 7,00

5° Análise e Conclusão:

A° Como é organizado um projeto corporativo no NetBeans?

R: A organização de um projeto corporativo no NetBeans, assim como em qualquer IDE, é fundamental para garantir a manutenibilidade, escalabilidade e colaboração entre os desenvolvedores.

Tamanho e complexidade do projeto: Projetos maiores tendem a necessitar de uma organização mais detalhada.

Linguagens e tecnologias utilizadas: Diferentes tecnologias podem influenciar a estrutura do projeto.

Convenções da empresa: Muitas empresas possuem padrões e diretrizes específicas para a organização de projetos.

Framework utilizado: Frameworks como Spring, Hibernate e JSF já possuem suas próprias convenções de organização.

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)

Persistência de dados: A JPA é responsável por mapear objetos Java para um banco de dados relacional. Ela abstrai os detalhes de acesso ao banco de dados, permitindo que os desenvolvedores se concentrem na lógica de negócio.

ORM (Object-Relational Mapping): A JPA utiliza o ORM para relacionar objetos Java com tabelas de um banco de dados. Isso facilita a manipulação de dados, pois os desenvolvedores podem trabalhar com objetos em vez de executar consultas SQL diretamente.

Annotations: As anotações JPA são usadas para definir o mapeamento entre classes Java e tabelas, relacionamentos entre entidades, e outras configurações de persistência.

JPQL (Java Persistence Query Language): A JPQL é uma linguagem de consulta orientada a objetos que permite realizar consultas sobre os dados persistidos.

EJB (Enterprise JavaBeans)

Lógica de negócio: Os EJBs são componentes Java que encapsulam a lógica de negócio de uma aplicação. Eles oferecem serviços como transações, segurança, e gerenciamento de estado.

Tipos de EJBs: Existem diferentes tipos de EJBs, como Session Beans (Stateless, Stateful), Message-Driven Beans e Entity Beans (obsoletos, substituídos pela JPA).

Container: Os EJBs são executados em um container EJB, que fornece um ambiente de execução gerenciado e oferece serviços adicionais.

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

R: O NetBeans, como uma IDE (Integrated Development Environment) robusta, oferece um conjunto de ferramentas e recursos que simplificam significativamente o desenvolvimento de aplicações Java utilizando JPA e EJB. Ao automatizar tarefas repetitivas e fornecer um ambiente visual intuitivo, o NetBeans contribui para uma maior produtividade dos desenvolvedores.

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 atuam como o coração de aplicações web dinâmicas. Eles são responsáveis por receber requisições HTTP, processar essas requisições e gerar respostas dinâmicas, geralmente na forma de páginas HTML.

Em resumo: Servlets são a ponte entre o mundo da web (requisições HTTP) e o mundo Java, permitindo a criação de aplicações web interativas e dinâmicas.

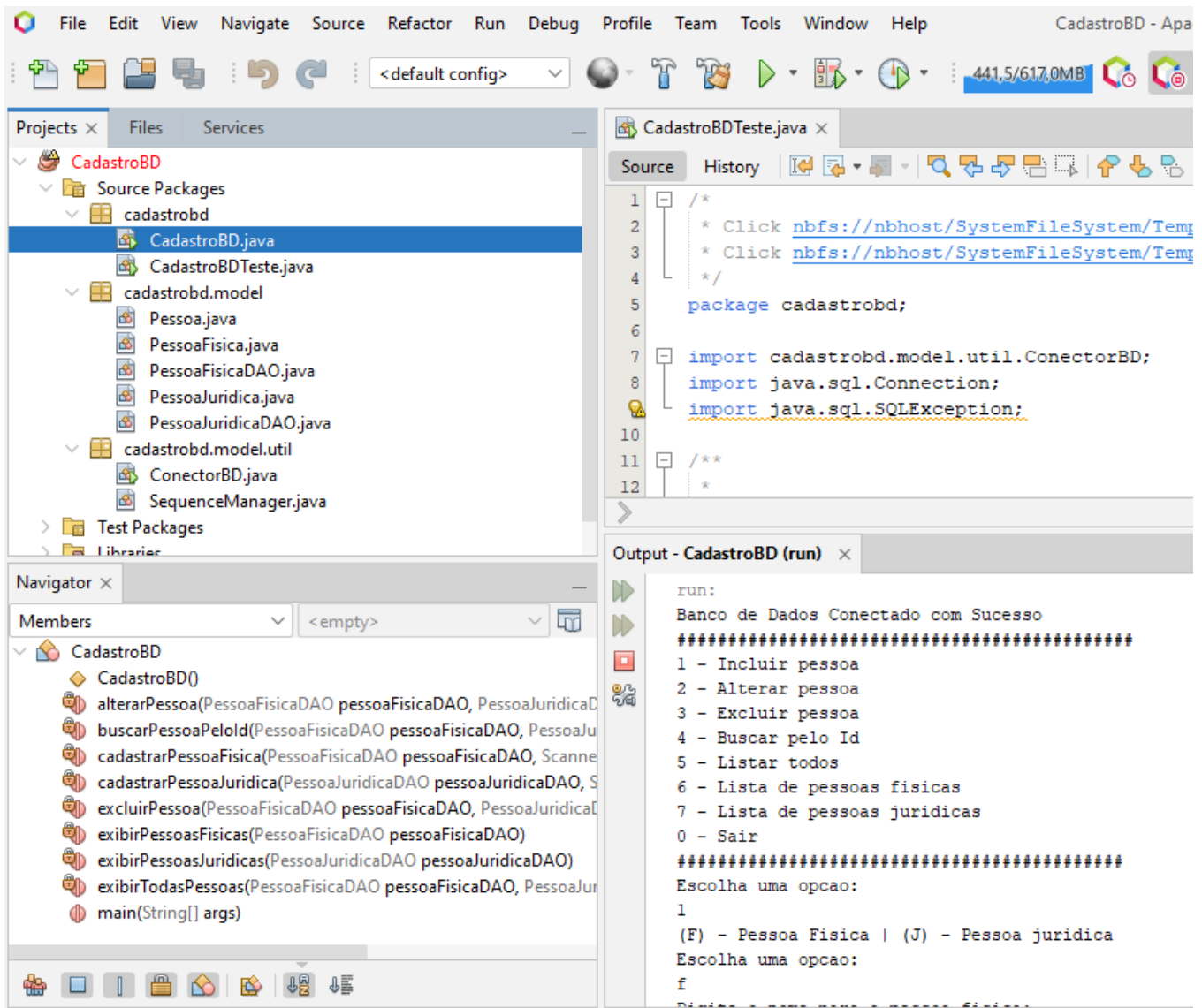
E° Como é feita a comunicação entre os Servlets e os Session Beans do pool de EJBs?

R: A comunicação entre Servlets e Session Beans é um aspecto fundamental nas aplicações Java EE. Essa interação permite que a camada de apresentação (Servlets) acesse a lógica de negócio encapsulada nos Session Beans, promovendo uma clara separação de responsabilidades e facilitando a manutenção e escalabilidade da aplicação.

1º Título da Prática: Interface Cadastral com Servlet e JSP

2º 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.



3º Códigos Solicitados:

3.1 Código do ServletProdutoFC.

```
package cadastroee.servlets;
```

```
import cadastroee.model.Produto;
import cadastroee.controller.ProdutoFacadeLocal;
import java.io.IOException;
import java.util.List;
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;
```

```

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

    @EJB
    private ProdutoFacadeLocal facade;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String acao = request.getParameter("acao");
        String destino = handleGetAction(acao, request);
        dispatchRequest(request, response, destino);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String acao = request.getParameter("acao");
        acao = acao == null || acao.isEmpty() ? " " : acao;
        String destino = handlePostAction(acao, request);
        dispatchRequest(request, response, destino);
    }

    private String handleGetAction(String acao, HttpServletRequest request) {
        switch (acao) {
            case "formIncluir":
                return "ProdutoDados.jsp";
            case "excluir":
                return handleExcluir(request);
            case "formAlterar":
                return handleFormAlterar(request);
            default:
                return handleListarProdutos(request);
        }
    }

    private String handlePostAction(String acao, HttpServletRequest request) {
        switch (acao) {
            case "incluir":
                return handleIncluir(request);
            case "alterar":
                return handleAlterar(request);
            default:
                return handleListarProdutos(request);
        }
    }

    private void dispatchRequest(HttpServletRequest request, HttpServletResponse response, String destino)
        throws ServletException, IOException {
        RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
    }
}

```

```

    dispatcher.forward(request, response);
}

// Métodos auxiliares para lidar com cada ação
private String handleExcluir(HttpServletRequest request) {
    int idDel = Integer.parseInt(request.getParameter("id"));
    facade.remove(facade.find(idDel));
    return handleListarProdutos(request);
}

private String handleFormAlterar(HttpServletRequest request) {
    int id = Integer.parseInt(request.getParameter("id"));
    Produto produto = facade.find(id);
    request.setAttribute("produto", produto);
    return "ProdutoDados.jsp";
}

private String handleListarProdutos(HttpServletRequest request) {
    List<Produto> produtos = facade.findAll();
    request.setAttribute("produtos", produtos);
    return "DbLista.jsp";
}

private String handleIncluir(HttpServletRequest request) {
    String nome = request.getParameter("nome");
    int quantidade = Integer.parseInt(request.getParameter("quantidade"));
    Float precoVenda = Float.valueOf(request.getParameter("precoVenda"));

    Produto newProduto = new Produto();
    newProduto.setNome(nome);
    newProduto.setQuantidade(quantidade);
    newProduto.setPrecoVenda(precoVenda);

    facade.create(newProduto);
    return handleListarProdutos(request);
}

private String handleAlterar(HttpServletRequest request) {
    Produto alterarProduto = facade.find(Integer.valueOf(request.getParameter("id")));

    String alterarNome = request.getParameter("nome");
    int alterarQuantidade = Integer.parseInt(request.getParameter("quantidade"));
    Float alterarPrecoVenda = Float.valueOf(request.getParameter("precoVenda"));

    alterarProduto.setNome(alterarNome);
    alterarProduto.setQuantidade(alterarQuantidade);
    alterarProduto.setPrecoVenda(alterarPrecoVenda);

    facade.edit(alterarProduto);
    return handleListarProdutos(request);
}

```


}

}

3.2 Código JSP responsável pela exibição da lista de produtos.

```
<html>
<head>
  <title>Listagem de Produtos</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-T3c6Coli6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">
  <style>
    body {
      background-color: #f4f4f4;
      font-family: 'Arial', sans-serif;
    }
    .header-section {
      background-color: #007bff;
      color: white;
      padding: 20px 0;
      margin-bottom: 30px;
    }
    .header-section h1 {
      margin: 0;
      font-size: 2.5rem;
    }
    .card {
      background-color: white;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .table thead {
      background-color: #007bff;
      color: white;
    }
    .table-striped tbody tr:nth-of-type(odd) {
      background-color: rgba(0,123,255,.1);
    }
    .btn-primary {
      background-color: #007bff;
      border-color: #007bff;
    }
    .btn-danger {
      background-color: #dc3545;
      border-color: #dc3545;
    }
    .btn-sm {
      padding: .25rem .5rem;
      font-size: .875rem;
    }
```

```

        line-height: 1.5;
        border-radius: .2rem;
    }
</style>
</head>
<body>
    <div class="container">
        <div class="header-section text-center">
            <h1>Listagem de Produtos</h1>
        </div>

        <div class="card">
            <div class="card-body">

                <table class="table table-striped table-bordered table-responsive">
                    <thead>
                        <tr class="table-dark">
                            <th>ID</th>
                            <th>Produto</th>
                            <th>Quantidade</th>
                            <th>Preço</th>
                            <th>Ações</th>
                        </tr>
                    </thead>

                    <%
                        DecimalFormat df = new DecimalFormat("#,##0.00");
                        List<Produto> produtos = (List<Produto>) request.getAttribute("produtos");

                        if (produtos != null && !produtos.isEmpty()) {
                            for (Produto produto : produtos) {
                                %>
                                <tr>
                                    <td class="text-center"><%=produto.getIdProduto()%></td>
                                    <td class="text-center"><%=produto.getNome()%></td>
                                    <td class="text-center"><%=produto.getQuantidade()%></td>
                                    <td class="text-center">R$ <%=df.format(produto.getPrecoVenda())%></td>
                                    <td class="text-end">
                                        <a class="btn btn-primary btn-sm"
href="ServletProdutoFC?acao=formAlterar&id=<%=produto.getIdProduto()%>">Alterar</a>
                                        <a class="btn btn-danger btn-sm"
href="ServletProdutoFC?acao=excluir&id=<%=produto.getIdProduto()%>">Excluir</a>
                                    </td>
                                </tr>
                            <%
                                }
                            } else {
                                %>
                                <tr>

```

```

        <td colspan="5">Nenhum produto encontrado.</td>
    </tr>
    <%
    }
    %>
</table>

<div class="text-end mb-3">
    <a class="btn btn-primary" href="ServletProdutoFC?acao=formIncluir">Cadastrar Produto</a>
</div>
</div>
</div>
</div>
</body>
</html>

```

4º Os resultados da execução dos códigos:

ServletProdutoFC em execução:



ID	Produto	Quantidade	Preço	Ações	
1	Banana	100	R\$ 5,00	Alterar	Excluir
2	Laranja	500	R\$ 2,00	Alterar	Excluir
3	Manga	800	R\$ 4,00	Alterar	Excluir
4	Tangerina	600	R\$ 7,00	Alterar	Excluir

[Cadastrar Produto](#)

5º Análise e Conclusão:

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

R: padrão Front Controller é um padrão de projeto arquitetural que centraliza o tratamento de todas as

requisições HTTP de uma aplicação web em um único ponto de entrada, geralmente um servlet. Esse servlet, chamado de front controller, é responsável por interceptar todas as requisições, determinar qual ação deve ser executada e delegar o processamento para outros componentes da aplicação.
Benefícios do Front Controller:

Centralização: Simplifica a manutenção e o gerenciamento da aplicação, pois a lógica de roteamento e tratamento de requisições está concentrada em um único ponto.

Reusabilidade: Permite a criação de componentes reutilizáveis para tratar diferentes tipos de requisições.

Facilidade de teste: Facilita a criação de testes unitários, pois a lógica de negócio está separada da interface com o usuário.

Segurança: Permite implementar mecanismos de segurança de forma centralizada, como autenticação e autorização.

B° Quais as diferenças e semelhanças entre Servlets e JSPs?

R: Uma Comparação Detalhada

Servlets e **JSPs** são tecnologias fundamentais no desenvolvimento de aplicações web Java, e embora trabalhem em conjunto, possuem características e finalidades distintas.

Servlets

Natureza: São classes Java que estendem a classe `HttpServlet`.

Função: Processam requisições HTTP e geram respostas dinâmicas.

Código: Contém a lógica de negócio da aplicação, como acesso a banco de dados, cálculos e manipulação de dados.

Saída: Geralmente geram conteúdo HTML de forma programática, utilizando `PrintWriter`.

Quando usar: Ideal para a lógica de negócio complexa, processamento de dados e interação com o banco de dados.

JSPs

Natureza: São páginas HTML com tags especiais (expressões e ações JSP).

Função: Geram conteúdo dinâmico misturando HTML estático com código Java.

Código: Contém principalmente HTML, com trechos de código Java embutidos para gerar conteúdo dinâmico.

Saída: Geram diretamente o HTML, mesclando o conteúdo estático com o dinâmico.

Quando usar: Ideal para a criação de interfaces de usuário, onde a maior parte do conteúdo é HTML e há apenas alguns elementos dinâmicos.

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: Redirecionamento Simples (sendRedirect)

O que acontece: O servidor envia uma resposta HTTP 302 (Found) para o cliente, informando que a página solicitada foi movida para um novo local. O navegador então faz uma nova requisição para o novo URL.

Quando usar:

Ao finalizar uma transação e direcionar o usuário para uma página de confirmação.

Ao redirecionar para uma página externa ao seu aplicativo.

Quando você quer que o navegador tenha um novo histórico de navegação.

R: Forward com RequestDispatcher

O que acontece: O servidor encaminha a requisição internamente para outro recurso (outro servlet, JSP, etc.), sem que o cliente seja notificado. O segundo recurso processa a requisição e gera a resposta.

Quando usar:

Para passar o controle para outro recurso dentro da mesma aplicação.

Quando você quer preservar os atributos da requisição original.

Quando você não quer que o navegador tenha um novo histórico de navegação.

1º Título da Prática: Melhorando o Design da Interface

2º 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

4º Os resultados da execução dos códigos:

4.1 ServletProdutoFC em execução, com uso de Bootstrap.



ID	Produto	Quantidade	Preço	Ações
1	Banana	100	R\$ 5,00	Alterar Excluir
2	Laranja	500	R\$ 2,00	Alterar Excluir
3	Manga	800	R\$ 4,00	Alterar Excluir
4	Tangerina	600	R\$ 7,00	Alterar Excluir

[Cadastrar Produto](#)

4.2 Cadastro de Produto



[Voltar](#)

Nome

Quantidade

Preço de Venda

[Cadastrar](#)

4.3 Alteração de Produto já cadastrado.



Alteração de Produto

Voltar

Nome
Tangerina

Quantidade
600

Preço de Venda
7.0

Alterar

5° Análise e Conclusão:

A° Como o framework Bootstrap é utilizado?

R: Bootstrap é amplamente utilizado para desenvolver interfaces de usuário responsivas, mobile first, para websites e aplicações web. Oferece um conjunto robusto de ferramentas baseadas em HTML, CSS e JavaScript, que incluem templates pré desenvolvidos para botões, formulários, navegação e outros elementos de interface, além de um sistema de grid flexível para layout.

B° Por que o Bootstrap garante a independência estrutural do HTML?

R: O Bootstrap assegura a independência estrutural do HTML, ao fornecer um conjunto de classes CSS pré-definidas, assim como componentes de interface que podem ser facilmente integrados ao HTML. Em razão disso, ao invés de escrever e ajustar diferentes estilos CSS personalizados para cada elemento, os desenvolvedores podem simplesmente usar as classes do Bootstrap para alcançar um design consistente e responsivo..

Em resumo:

O Bootstrap é uma ferramenta poderosa para criar layouts responsivos e consistentes, mas não garante uma independência estrutural completa do HTML. Para alcançar um maior desacoplamento, você pode combinar o Bootstrap com outras técnicas e ferramentas que promovam uma separação mais clara entre a estrutura, a apresentação e o comportamento.

C° Qual a relação entre o Bootstrap e a responsividade da página?

R: O Bootstrap simplifica enormemente o processo de criação de sites responsivos, oferecendo um conjunto de ferramentas e componentes pré-construídos que facilitam a adaptação do layout da sua página a diferentes dispositivos. Ao utilizar o Bootstrap, você ganha tempo e evita ter que escrever grandes quantidades de CSS personalizado para garantir a responsividade do seu site.