

Procedimento 2 - Interface Cadastral com Servlet e JSPs

Objetivos

O principal objetivo deste procedimento foi desenvolver uma aplicação web funcional baseada na plataforma Java EE, com foco na implementação de um CRUD (Create, Read, Update, Delete) para o gerenciamento completo da entidade **Produto**. A proposta buscou exercitar conhecimentos de desenvolvimento Full Stack utilizando tecnologias padrão do ecossistema Java corporativo, promovendo a integração entre camadas de apresentação, controle e persistência de dados.

O foco foi explorar a integração entre Servlets, JSP, EJB e JPA em um ambiente web.

Tecnologias Utilizadas

  Jakarta EE

 Servlets

 EJB (Enterprise JavaBeans)

 JPA (Jakarta Persistence API)

 SQL Server

 JDBC Driver para SQL Server (configurado no GlassFish)

 NetBeans

 GlassFish

 Git


 Java

 JSP

 HTML

 Ant (sistema de build)

 JDK 17

 Google Chrome



Estrutura do Projeto

O projeto foi dividido nas seguintes camadas:

- modelo/Produto.java → Classe entidade representando a tabela produto, anotada com JPA para mapeamento objeto-relacional.
- modelo/ProdutoDAO.java → Classe de acesso a dados usando JPA.
- Interface ProdutoFacadeLocal encapsulando o acesso aos dados com EJB.
- controle/ServletProdutoFC.java → *Servlet* controlador responsável por gerenciar as requisições CRUD para inclusão, listagem, alteração e exclusão e encaminhar para as páginas JSP.
- ProdutoDados.jsp → Formulário para inclusão e edição de produtos.
- ProdutoLista.jsp → Página para listagem e gerenciamento de produtos com opções de alteração e exclusão.



Funcionalidades

- Cadastro de novos produtos com nome, quantidade e preço.
- Listagem de todos os produtos em formato de tabela.
- Edição dos dados de um produto existente.
- Exclusão de produtos com atualização automática da lista.
- Links dinâmicos para ações no ServletProdutoFC.
- Encaminhamento de requisições por RequestDispatcher.



Atividades Realizadas

- ✓ Criação da estrutura de diretórios Java EE.
- ✓ Modelagem da entidade Produto.
- ✓ Implementação da camada de persistência com JPA.
- ✓ Desenvolvimento do servlet controlador com tratamento de múltiplas ações (acao).
- ✓ Criação das páginas JSP para interação com o usuário.
- ✓ Testes funcionais de cada operação do CRUD.
- ✓ Tratamento de conversão de tipos (ex: Float.parseFloat para preços).
- ✓ Organização de navegação entre páginas e envio de dados.

Desafios Enfrentados

- ! Conversão correta de valores numéricos no `request.getParameter` para tipos primitivos como `float` e `int`.
- ! Controle de redirecionamento e envio de parâmetros entre páginas JSP e Servlets.
- ! Configuração do ambiente de servidor (GlassFish/WildFly) e persistência com JPA.
- ! Tratamento de requisições com múltiplas ações no servlet usando `if-else`.
- ! Evitar erros de `NullPointerException` ao acessar objetos ou listas não inicializadas.

Códigos

Os códigos foram desenvolvidos com a IDE NetBeans e se encontram em repositório no GitHub onde podem ser acessados pelo link: https://github.com/CarlosCatao/Mundo_3_Nivel_4_Missao_Pratica/tree/main/Procedimento-2/CadastroEE.

Testes

O processo de testes é essencial para garantir a robustez do sistema e identificar pontos de melhoria tanto na interface quanto na lógica de negócios. Através dessas iterações, o sistema evolui para uma versão estável e funcional.

Testes Realizados

1. Teste de Inclusão

- Foram testados cenários com campos válidos e inválidos. Verificou-se a adição correta do produto na lista e a persistência dos dados no banco.

2. Teste de Alteração

- Verificou-se se os dados do produto eram carregados corretamente ao clicar em "Alterar", e se, após a edição, os dados eram atualizados corretamente na listagem.

3. Teste de Exclusão

- Testou-se a exclusão de produtos com e sem confirmação. Após confirmar a exclusão, foi verificado se o produto desaparecia da lista.

4. Teste de Validação

- Foram realizados testes para envio do formulário com campos em branco, valores negativos e preço com formato inválido.

5. Teste de Navegação

- Todos os fluxos de navegação foram testados: incluir, alterar, excluir, voltar à lista. A consistência do estado da aplicação foi verificada após cada ação.

Resultados

Os resultados da execução dos códigos se encontram ilustrados no arquivo *Resultados.pdf* que se encontra em repositório no GitHub onde podem ser acessados pelo link:

https://github.com/CarlosCatao/Mundo_3_Nivel_4_Missao_Pratica/blob/main/Procedimento-2/RESULTADOS.pdf.

Conclusão

O desenvolvimento deste procedimento permitiu aplicar os principais conceitos da arquitetura Java EE para web, incluindo a separação de responsabilidades entre camada de apresentação, controle e persistência. A integração entre Servlets e JSP foi consolidada, assim como a utilização de JPA para abstração do banco de dados.

O uso de ServletProdutoFC como controlador central demonstrou ser uma abordagem eficaz para controlar a lógica do sistema e a navegação entre as telas.

Como resultado final, obteve-se uma aplicação funcional, organizada e preparada para futuras extensões, como autenticação de usuários ou filtros por categorias.

O projeto reforçou a importância de boas práticas no desenvolvimento Full Stack, como reutilização de código, clareza na organização e validação de dados.

? Questionamentos

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

O padrão **Front Controller** é um padrão de projeto onde todas as requisições passam por um único ponto de entrada, que centraliza o controle da navegação e do fluxo da aplicação.

- Recebe todas as requisições HTTP.
- Analisa a requisição (parâmetros, URL, ações).
- Encaminha a requisição para o componente apropriado (geralmente um Controller específico ou diretamente para a View).
- É o coração da camada **Controller** na arquitetura MVC.

Implementação em Java:

Em aplicações Java Web, o Front Controller, geralmente, é implementado como um **Servlet** ou com frameworks como **Spring MVC**, que automatizam isso com controllers anotados.

▶ Quais as diferenças e semelhanças entre Servlets e JSPs?

Diferenças:

Característica	Servlet	JSP (JavaServer Pages)
Objetivo principal	Lógica de controle (Controller)	Apresentação (View)
Sintaxe	Código Java puro	Mistura HTML com Java (scripts ou EL)
Foco	Processamento de requisições	Renderização da resposta (HTML)
Facilidade	Menos legível para interface	Mais intuitivo para designers
Compilação	Já compilado como classe Java	Traduzido em Servlet na primeira execução

Semelhanças:

- Ambos são executados no servidor e geram conteúdo dinâmico.
- Fazem parte da especificação Java EE.
- Podem acessar `HttpServletRequest`, `HttpServletResponse`, sessões e contexto da aplicação.

- Podem se comunicar entre si.

Finalizando, pode-se afirmar que **Servlets** controlam o fluxo e **JSPs** exibem a interface.

► 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 HttpServletRequest?

Redirecionamento Simples (response.sendRedirect):

- Envia uma nova resposta HTTP com código 302 (redirecionamento).
- O navegador faz nova requisição para a nova URL.
- Perde os dados do *request* original.
- A URL no navegador é atualizada.
- Útil para redirecionar para outra página após um **POST**, por exemplo.

Encaminhamento (RequestDispatcher.forward):

- Ocorre internamente no servidor, sem o navegador saber.
- A requisição não é reiniciada, os objetos *request* e *response* são mantidos.
- A URL no navegador permanece a mesma.
- Útil para MVC: *controller* processa e encaminha para JSP exibir.

Parâmetros vs. Atributos no HttpServletRequest:

Tipo	Método	Escopo	Exemplo de uso
Parâmetro	request.getParameter()	Requisição HTTP (query/form)	Dados enviados pelo usuário
Atributo	request.setAttribute()	Durante o ciclo da requisição	Dados entre Servlet e JSP