Campus: Asa Norte

Curso: Desenvolvimento Full Stack

Disciplina: Vamos Integrar Sistemas

Matrícula: 2023.09.96862-2

Semestre Letivo: 3º Semestre

Integrantes: André Luis Soares de Oliveira

Desenvolvimento de Aplicação Web Java: Integração de JPA, EJB e Servlets no Ambiente NetBeans

Objetivo da Prática

Desenvolver uma aplicação web utilizando JPA, EJB e Servlets no NetBeans, visando:

1. Compreender a arquitetura de aplicações Java, integrando as camadas de apresentação, negócio e persistência.
2. Aprender a implementar funcionalidades de gerenciamento de dados e lógica de negócios.
3. Experienciar a integração das tecnologias JPA e EJB para manipulação eficiente de dados.
4. Familiarizar-se com o ambiente de desenvolvimento NetBeans e suas ferramentas para aumentar a produtividade.
5. Praticar a comunicação entre Servlets e EJBs, promovendo a integração das diferentes camadas da aplicação.

Códigos Solicitados

Classe de entidade JPA:

```
import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;


@Entity

public class Produto {


  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  private Long id;
```

```
private String nome;
private Double preco;

// Construtor padrão
public Produto() {}

// Construtor com parâmetros
public Produto(String nome, Double preco) {
   this.nome = nome;
   this.preco = preco;
}

// Getters e Setters
public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public String getNome() {
   return nome;
}

public void setNome(String nome) {
```

```
        this.nome = nome;
    }

    public Double getPreco() {
        return preco;
    }

    public void setPreco(Double preco) {
        this.preco = preco;
    }
}
```

Session Bean EJB:

```
import jakarta.ejb.Stateless;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import java.util.List;

@Stateless
public class ProdutoServiceBean {

    @PersistenceContext(unitName = "MeuPU")
    private EntityManager em;

    // Método para adicionar um novo produto
    public void adicionarProduto(Produto produto) {
```

```
        em.persist(produto);
    }

    // Método para buscar um produto pelo ID
    public Produto buscarProduto(Long id) {
        return em.find(Produto.class, id);
    }

    // Método para listar todos os produtos
    public List<Produto> listarProdutos() {
        return em.createQuery("SELECT p FROM Produto p",
Produto.class).getResultList();
    }
}
```

Servlet que interage com o session bean:

```
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 jakarta.ejb.EJB;
import java.io.IOException;

@WebServlet("/produto")
public class ProdutoServlet extends HttpServlet {
```

```
    @EJB
    private ProdutoServiceBean produtoService;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String nome = request.getParameter("nome");
        Double preco = Double.parseDouble(request.getParameter("preco"));

        Produto novoProduto = new Produto(nome, preco);
        produtoService.adicionarProduto(novoProduto);

        response.sendRedirect("produtos");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        request.setAttribute("produtos", produtoService.listarProdutos());
        request.getRequestDispatcher("/listaProdutos.jsp").forward(request, response);
    }
}
```

Página JSP para exibir os produtos:

```
<%@ page contentType="text/html;charset=UTF-8" %>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista de Produtos</title>
</head>
<body>
  <h1>Produtos Disponíveis</h1>
  <table border="1">
    <tr>
      <th>ID</th>
      <th>Nome</th>
      <th>Preço</th>
    </tr>
    <c:forEach var="produto" items="${produtos}">
      <tr>
        <td>${produto.id}</td>
        <td>${produto.nome}</td>
        <td>${produto.preco}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

Arquivo de configuração persistence.xml:

```
<persistence xmlns="http://jakarta.ee/xml/ns/persistence" version="3.0">
```

```
  <persistence-unit name="MeuPU" transaction-type="JTA">
    <jta-data-source>java:comp/DefaultDataSource</jta-data-source>
    <properties>
      <property name="jakarta.persistence.schema-generation.database.action"
value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Análise e Conclusão

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 design que centraliza o tratamento das requisições em um único ponto, o que facilita a gestão do fluxo de controle da aplicação. Em um aplicativo Web Java, isso é tipicamente implementado através de um Servlet que atua como o controlador principal. Quando uma requisição é recebida, o Front Controller decide qual lógica de processamento deve ser aplicada com base na URL da requisição e direciona a chamada para o controlador apropriado, que pode ser uma classe Java que implementa a lógica do negócio ou outro recurso. Esse padrão permite a separação de preocupações, uma vez que o Servlet front controller manipula a lógica de navegação enquanto a lógica de apresentação pode ser delegada para JSPs ou outros mecanismos de visualização.

Quais as diferenças e semelhanças entre Servlets e JSPs?

Servlets e JSPs são ambas tecnologias utilizadas na construção de aplicações web em Java, mas possuem propósitos e modos de operação diferentes.

- **Servelts:** São classes Java que lidam com a lógica de negócios e o processamento de requisições. Eles são mais adequados para situações que exigem manipulação complexa de dados e interação com a lógica de backend. Servlets geram conteúdo dinâmico respondendo a requisições HTTP.
- **JSPs (JavaServer Pages):** São uma forma simplificada de criar conteúdo dinâmico. Elas permitem a inclusão de código Java em uma página HTML, sendo mais focadas na apresentação de dados. JSPs são compiladas em Servlets pelo servidor, o que

significa que são convertidas em classes Java que podem ser manipuladas como Servlets.

Semelhanças: Ambos podem ser utilizados juntos em um aplicativo web, e ambos seguem o mesmo ciclo de vida de uma requisição HTTP. Ambos também podem interagir com objetos de requisição e resposta para manipular dados.

Qual a diferença entre um redirecionamento simples e o uso do método forward, a partir do RequestDispatcher?

- **Redirecionamento:** Quando um redirecionamento é realizado, o servidor envia uma resposta HTTP com um status de redirecionamento (geralmente 302) para o cliente, que então faz uma nova requisição para a URL especificada. Isso resulta em duas requisições HTTP e a URL exibida no navegador é a do recurso para o qual o redirecionamento foi feito.
- **Forward:** O método forward do `RequestDispatcher` é utilizado para encaminhar a requisição para outro recurso no servidor (como um Servlet ou uma JSP) sem que o cliente tenha conhecimento disso. Nesse caso, apenas uma requisição HTTP ocorre, e a URL no navegador não muda. A requisição e a resposta permanecem no servidor.

Para que servem parâmetros e atributos nos objetos HttpRequest?

- **Parâmetros:** São utilizados para passar dados de um cliente (como informações de um formulário) para o servidor. Esses dados podem ser acessados através do método `getParameter()` do objeto `HttpServletRequest`. Parâmetros são típicos em requisições GET e POST e são utilizados principalmente para obter informações que o usuário inseriu.
- **Atributos:** São utilizados para armazenar informações temporárias que podem ser compartilhadas entre Servlets e JSPs durante o processamento de uma única requisição. Atributos são definidos através do método `setAttribute()` do objeto `HttpServletRequest` e podem ser acessados através do método `getAttribute()`. Eles são úteis para transferir dados entre diferentes componentes da aplicação sem a necessidade de expô-los como parâmetros de URL ou formulários.

Link Github: https://github.com/andreluissdo/Missao-Pratica-4.git