Vamos Integrar Sistemas

João Pedro da Silva Zanirati Nunes – 202209081405
Polo Azenha
Nível 4: Vamos integrar sistemas – 9003 – Mundo 3

Objetivo da Prática

O objetivo é criar uma aplicação web usando Java EE, onde utiliza um bancos de dados, constroi interfaces de usuário interativas, e aplica boas práticas de desenvolvimento.

1º Procedimento | Camadas de Persistência e Controle

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.1" 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_1.xsd">
  <persistence-unit name="CadastroEE-ejbPU" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>cadastroee.model.Pessoa</class>
    <class>cadastroee.model.PessoaFisica</class>
    <class>cadastroee.model.Produto</class>
    <class>cadastroee.model.ProdutoMovimento</class>
    <class>cadastroee.model.Usuario</class>
    <jta-data-source>jdbc/loja</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
<property name="javax.persistence.jdbc.url"
value="jdbc:sqlserver://localhost:1433;databaseName=LOJA"/>
```

```
      <property name="javax.persistence.jdbc.user" value="loja"/>
      <property name="javax.persistence.jdbc.password" value="loja"/>
                                          <property        name="javax.persistence.jdbc.driver"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
    </properties>
  </persistence-unit>
</persistence>


package cadastroee.model;


import java.io.Serializable;
import java.util.Collection;
import jakarta.persistence.Basic;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.NamedQueries;
import jakarta.persistence.NamedQuery;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;


@Entity
```

```
@Table(name = "produto", catalog = "LOJA", schema = "LOJA")
@NamedQueries({
    @NamedQuery(name = "Produto_1.findAll", query = "SELECT p FROM Produto_1 p"),
    @NamedQuery(name = "Produto_1.findByIdProduto", query = "SELECT p FROM Produto_1 p WHERE p.idProduto = :idProduto"),
    @NamedQuery(name = "Produto_1.findByNomeProduto", query = "SELECT p FROM Produto_1 p WHERE p.nomeProduto = :nomeProduto"),
    @NamedQuery(name = "Produto_1.findByQtd", query = "SELECT p FROM Produto_1 p WHERE p.qtd = :qtd"),
    @NamedQuery(name = "Produto_1.findByValorUnitario", query = "SELECT p FROM Produto_1 p WHERE p.valorUnitario = :valorUnitario"),
    @NamedQuery(name = "Produto_1.findByTipo", query = "SELECT p FROM Produto_1 p WHERE p.tipo = :tipo")})
public class Produto implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "idProduto")
    private Integer idProduto;
    @Basic(optional = false)
    @Column(name = "nomeProduto")
    private String nomeProduto;
    @Basic(optional = false)
    @Column(name = "qtd")
    private int qtd;
```

```
    @Basic(optional = false)
    @Column(name = "valorUnitario")
    private float valorUnitario;
    @Basic(optional = false)
    @Column(name = "tipo")
    private String tipo;
    @JoinColumn(name = "idJuridica", referencedColumnName = "idJuridica")
    @ManyToOne(optional = false)
    private PessoaJuridica idJuridica;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idProduto")
    private Collection<ProdutoMovimento> produtoMovimentoCollection;

    public Produto() {
    }
    public Produto(Integer idProduto) {
        this.idProduto = idProduto;
    }
     public Produto(Integer idProduto, String nomeProduto, int qtd, long valorUnitario,
String tipo) {
        this.idProduto = idProduto;
        this.nomeProduto = nomeProduto;
        this.qtd = qtd;
        this.valorUnitario = valorUnitario;
        this.tipo = tipo;
    }
```

```
public Integer getIdProduto() {
    return idProduto;
}
public void setIdProduto(Integer idProduto) {
    this.idProduto = idProduto;
}
public String getNomeProduto() {
    return nomeProduto;
}
public void setNomeProduto(String nomeProduto) {
    this.nomeProduto = nomeProduto;
}
public int getQtd() {
    return qtd;
}
public void setQtd(int qtd) {
    this.qtd = qtd;
}
public float getValorUnitario() {
    return valorUnitario;
}
public void setValorUnitario(long valorUnitario) {
    this.valorUnitario = valorUnitario;
}
public String getTipo() {
```

```
        return tipo;
    }
    public void setTipo(String tipo) {
        this.tipo = tipo;
    }
    public PessoaJuridica getIdJuridica() {
        return idJuridica;
    }

    public void setIdJuridica(PessoaJuridica idJuridica) {
        this.idJuridica = idJuridica;
    }
    public Collection<ProdutoMovimento> getProdutoMovimentoCollection() {
        return produtoMovimentoCollection;
    }
        public void setProdutoMovimentoCollection(Collection<ProdutoMovimento>
produtoMovimentoCollection) {
        this.produtoMovimentoCollection = produtoMovimentoCollection;
    }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idProduto != null ? idProduto.hashCode() : 0);
        return hash;
    }
```

```
    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Produto)) {
            return false;
        }
        Produto other = (Produto) object;
          if ((this.idProduto == null && other.idProduto != null) || (this.idProduto != null
&& !this.idProduto.equals(other.idProduto))) {
            return false;
        }
        return true;
    }
    @Override
    public String toString() {
        return "cadastroee.model.Produto_1[ idProduto=" + idProduto + " ]";
    }
}


package cadastroee.servlets;


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;

public class ServletProduto extends HttpServlet {
   @EJB
   ProdutoFacadeLocal facade;

      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         /* TODO output your page here. You may use following sample code. */
         out.println("<!DOCTYPE html>");
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet ServletProduto</title>");
         out.println("</head>");
         out.println("<body>");
               out.println("<h1>Servlet ServletProduto at " + request.getContextPath() + "</h1>");
         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";
    }
}
```

Como é organizado um projeto corporativo no NetBeans?

Pastas Principais:

- nbproject: Configurações NetBeans.
- src: Código-fonte.
- lib: Bibliotecas externas.
- build: Arquivos compilados.
- dist: Artefato final.

Pacotes:

Organização lógica do código em pacotes.

Bibliotecas:

Gerenciamento de dependências externas.

Persistência:

Pacotes para entidades e controle JPA.

Interface do Usuário:

Pacotes para a camada de apresentação (JSF, Servlets, etc.).

Configurações e Recursos:

Armazenamento de configurações e recursos web.

Testes:

Seção para testes unitários.

Documentação:

Pasta ou pacote para documentação.

Versionamento:

Integração com sistemas de controle de versão.

Recursos Web:

Pasta para arquivos web (JSP, CSS, imagens).

Qual o papel das tecnologias JPA e EJB na construção de um aplicativo

para a plataforma Web no ambiente Java?

JPA: Mapeamento objeto-relacional para interação com bancos de dados.

EJB: Desenvolvimento de componentes empresariais reutilizáveis.

Integração em Aplicações Web:

JPA: Persistência de dados.

EJB: Lógica de negócios e operações empresariais em aplicativos web Java.

Como o NetBeans viabiliza a melhoria de produtividade ao lidar com as

tecnologias JPA e EJB?

O NetBeans oferece uma integração eficiente e assistência ao desenvolvedor para tecnologias JPA e EJB, facilitando tarefas como geração de código, configuração, depuração e criação de interfaces gráficas. Isso resulta em maior produtividade durante o desenvolvimento de aplicativos Java EE.

O que são Servlets, e como o NetBeans oferece suporte à construção

desse tipo de componentes em um projeto Web?

O NetBeans simplifica a criação de Servlets em projetos web, oferecendo modelos e assistentes para configuração, mapeamento e implementação desses componentes. facilitando o desenvolvimento e teste de Servlets no ambiente de desenvolvimento.

Como é feita a comunicação entre os Serlvets e os Session Beans do

pool de EJBs?

2º Procedimento | Interface Cadastral com Servlet e JSPs

```
package cadastroee.servlets;


import cadastroee.model.Produto;
import jakarta.ejb.EJB;
import jakarta.servlet.RequestDispatcher;
import java.io.IOException;
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.util.List;


@WebServlet("/ServletProdutoFC")
public class ServletProdutoFC extends HttpServlet {
   @EJB
   private ProdutoFacadeLocal facade;
      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      String acao = request.getParameter("acao");
      String destino = "";
      if (acao.equals("listar")) {
         List<Produto> listaProdutos = facade.listarProdutos();
         request.setAttribute("listaProdutos", listaProdutos);
         destino = "ProdutoLista.jsp";
      } else if (acao.equals("formIncluir")) {
         destino = "ProdutoDados.jsp";
      } else if (acao.equals("formAlterar")) {
         int id = Integer.parseInt(request.getParameter("id"));
         Produto produto = facade.buscarProdutoPorId(id);
         request.setAttribute("produto", produto);
         destino = "ProdutoDados.jsp";
      } else if (acao.equals("excluir")) {
         int id = Integer.parseInt(request.getParameter("id"));
```

```
            facade.removerProduto(id);
            List<Produto> listaProdutos = facade.listarProdutos();
            request.setAttribute("listaProdutos", listaProdutos);
            destino = "ProdutoLista.jsp";
        } else if (acao.equals("alterar")) {
            int id = Integer.parseInt(request.getParameter("id"));
            Produto produto = facade.buscarProdutoPorId(id);
            List<Produto> listaProdutos = facade.listarProdutos();
            request.setAttribute("listaProdutos", listaProdutos);
            destino = "ProdutoLista.jsp";
        } else if (acao.equals("incluir")) {
            Produto novoProduto = new Produto();
            facade.inserirProduto(novoProduto);
            List<Produto> listaProdutos = facade.listarProdutos();
            request.setAttribute("listaProdutos", listaProdutos);
            destino = "ProdutoLista.jsp";
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
        dispatcher.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);
    }
}
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="ISO-8859-1">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
                                                                    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet"                                                integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/
Dwwykc2MPK8M2HN" crossorigin="anonymous">
                                                                    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o
0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>
    </head>
    <body class="container">
    <h1>Lista de Produtos</h1>
```

```
    <a href="ServletProdutoFC?acao=formIncluir" class="btn btn-primary m-2">Incluir Produto</a>


   <table class="table table-striped">
      <thead class="thead-dark">
         <tr>
            <th>ID</th>
            <th>Nome</th>
            <th>Quantidade</th>
            <th>Preço</th>
            <th>Ações</th>
         </tr>
      </thead>
      <tbody>
         <c:forEach var="produto" items="${listaProdutos}">
            <tr>
               <td>${produto.id}</td>
               <td>${produto.nome}</td>
               <td>${produto.quantidade}</td>
               <td>${produto.preco}</td>
               <td>
                        <a href="ServletProdutoFC?acao=formAlterar&id=${produto.id}" class="btn btn-primary btn-sm">Alterar</a>
                     <a href="ServletProdutoFC?acao=excluir&id=${produto.id}" class="btn btn-danger btn-sm">Excluir</a>
               </td>
```

```
            </tr>
          </c:forEach>
        </tbody>
    </table>
</body>
</html>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
    <title>Dados do Produto</title>
     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet"                                                    integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/
Dwwykc2MPK8M2HN" crossorigin="anonymous">
                                                                    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o
0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>
</head>
<body class="container">
    <h1>Dados do Produto</h1>
    <form action="ServletProdutoFC" method="post" class="form">
            <input type="hidden" name="acao" value="${empty produto ? 'incluir' :
'alterar'}">
        <c:if test="${not empty produto}">
```

<input type="hidden" name="id" value="${produto.id}">

</c:if>

<label for="nome" class="form-label">Nome do Produto:</label>

<input type="text" id="nome" name="nome" class="form-control" value="${empty produto ? '' : produto.nome}" required>

<label for="quantidade" class="form-label">Quantidade:</label>

<input type="number" id="quantidade" name="quantidade" class="form-control" value="${empty produto ? '' : produto.quantidade}" required>

<label for="preco" class="form-label">Preço:</label>

<input type="text" id="preco" name="preco" class="form-control" value="${empty produto ? '' : produto.preco}" required>

<button type="submit" class="btn btn-primary">Salvar</button>

</form>

</body>

</html>

Como funciona o padrão Front Controller, e como ele é implementado em

um aplicativo Web Java, na arquitetura MVC?

O padrão Front Controller é uma abordagem de design que centraliza o processamento de solicitações em um componente principal, geralmente um servlet, que direciona as requisições para os controladores apropriados. No contexto da arquitetura MVC,

coordena a interação entre o modelo, a visão e o controle. Isso proporciona uma estrutura organizada para o desenvolvimento de aplicativos web, melhorando a manutenção e a clareza da lógica de controle.

Quais as diferenças e semelhanças entre Servlets e JSPs?

Semelhanças:

Ambos baseados em Java.

Permitem criar páginas web dinâmicas.

Diferenças:

Servlets:

Classes em Java para manipular solicitações HTTP.

Orientados a controladores.

Abstração mais baixa, manipulação direta do ciclo de vida da solicitação HTTP.

JSPs:

Páginas HTML com código Java incorporado.

Orientados à visão.

Abstração mais alta, facilita integração de código Java com HTML.

Diferenças em modelo de programação, abstração, manutenção, facilidade de desenvolvimento, reusabilidade.

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 Simples:

Cliente é enviado para uma nova URL, resultando em uma nova solicitação.

Forward com RequestDispatcher:

Requisição é despachada internamente no servidor, sem nova solicitação.

Útil para encaminhar a solicitação para outro recurso no mesmo aplicativo.

Parâmetros em HttpRequest:

Informações na URL, como dados de formulário em uma solicitação GET.

Atributos em HttpRequest:

Informações compartilhadas entre servlets ou partes do mesmo servlet.

Útil ao encaminhar solicitações, pois os atributos são passados para o recurso encaminhado.

3º Procedimento | Melhorando o Design da Interface

Como o framework Bootstrap é utilizado?

O Bootstrap é um framework utilizado para criar interfaces web responsivas, oferecendo estilos pré-definidos, componentes prontos e scripts JavaScript. Facilitando a criação de layouts adaptáveis, fornece componentes comumente utilizados e permitindo personalização do site de forma simples.

Por que o Bootstrap garante a independência estrutural do HTML?

O Bootstrap proporciona independência estrutural ao separar a estilização do HTML. Ele utiliza classes CSS predefinidas para aplicar estilos, permitindo que os desenvolvedores foquem na estrutura semântica do HTML. Isso facilita a manutenção, alteração visual e torna o código mais modular.

Qual a relação entre o Boostrap e a responsividade da página?

O Bootstrap é um framework que facilita a criação de páginas web responsivas, garantindo uma experiência consistente em diversos dispositivos, desde smartphones até desktops. Ele oferece um sistema de grid responsivo e classes CSS específicas para ajudar na adaptação ao tamanho da tela.

Conclusão

O projeto envolveu a criação de um aplicativo corporativo em Java, destacando o uso de tecnologias como JPA, EJB e Bootstrap. O NetBeans facilitou o desenvolvimento, oferecendo suporte integrado e agilizando tarefas. A prática explorou a interação entre Servlets, Session Beans e a configuração de persistência. A inclusão do Bootstrap melhorou a apresentação visual das páginas. Em resumo, a experiência proporcionou uma compreensão prática e integrada das ferramentas e tecnologias essenciais para aplicativos web Java corporativos.