



Estácio

FACULDADE ESTÁCIO

CÂMPUS VOLTA REDONDA – RJ

DESENVOLVIMENTO FULL STACK

DISCIPLINA – VAMOS INTEGRAR SISTEMAS

TURMA – 2023.2

SEMESTRE – 3

**VOLTA REDONDA, AGOSTO 2024.
DESENVOLVIMENTO FULL STACK**

DISCIPLINA – VAMOS INTEGRAR SISTEMAS

TURMA – 2023.2

SEMESTRE – 3

ALUNO – BRUNO SAMPAIO BASTOS

TUTOR – JHONATAN ALVES

GITHUB - <https://github.com/BrunoTI-Code?tab=repositories>

VOLTA REDONDA, AGOSTO 2024.

1 1º PROCEDIMENTO | CAMADAS DE PERSISTÊNCIA E CONTROLE

1.1 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.

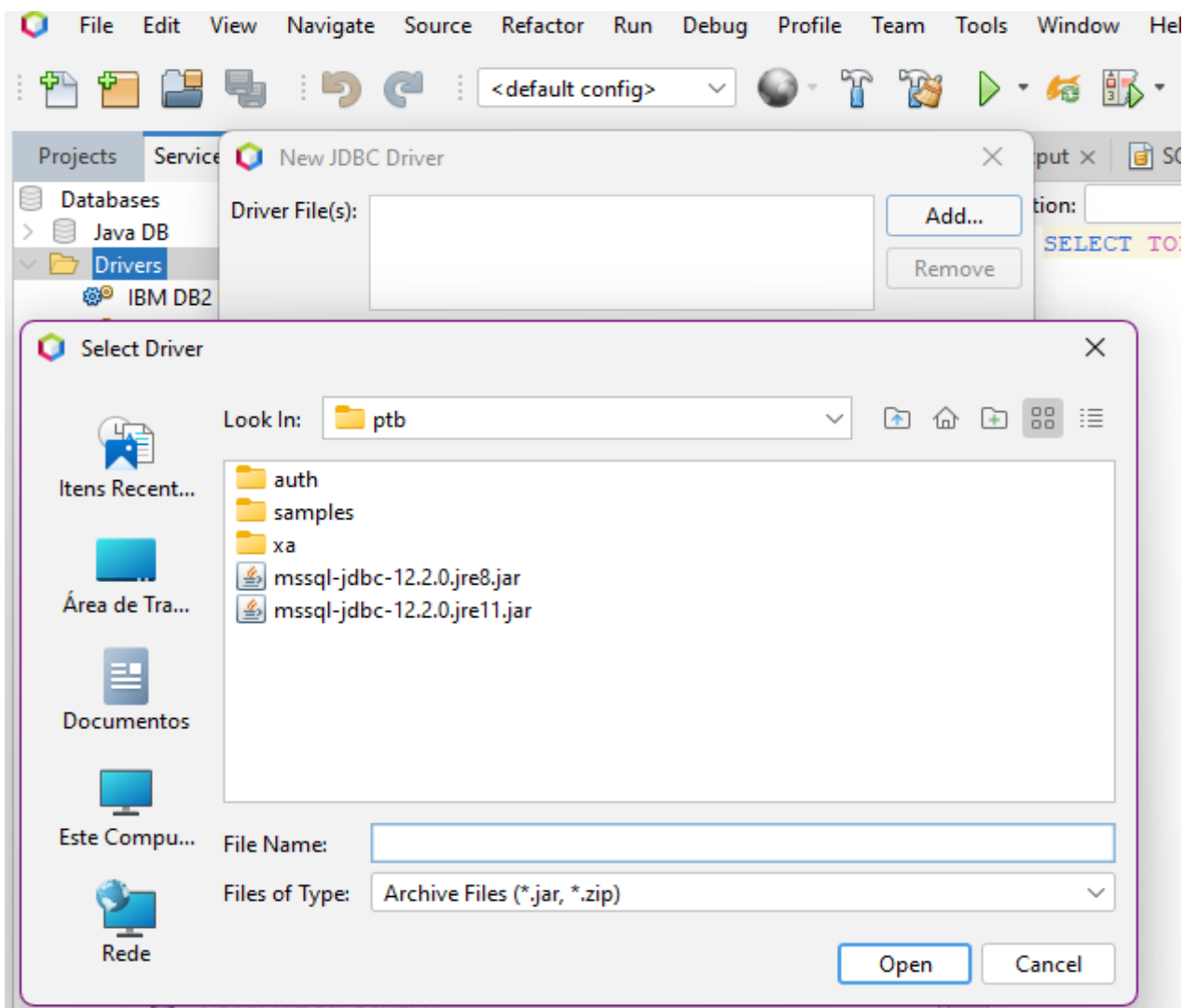
Utilizar a biblioteca Bootstrap para melhoria do design.

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.

2 CONFIGURAR A CONEXÃO COM SQL SERVER VIA NETBEANS E O POOL DE CONEXÕES NO GLASSFISH SERVER 6.2.1

- Na aba de Serviços, divisão **Banco** de Dados, clique com o botão direito em Drivers e escolha Novo Driver.
- Na janela que se abrirá, clicar em Add (Adicionar), selecionar o arquivo mssql-jdbc-12.2.0.jre8.jar, que é parte do arquivo zip encontrado no endereço seguinte, e finalizar com Ok

<https://learn.microsoft.com/pt-br/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16>



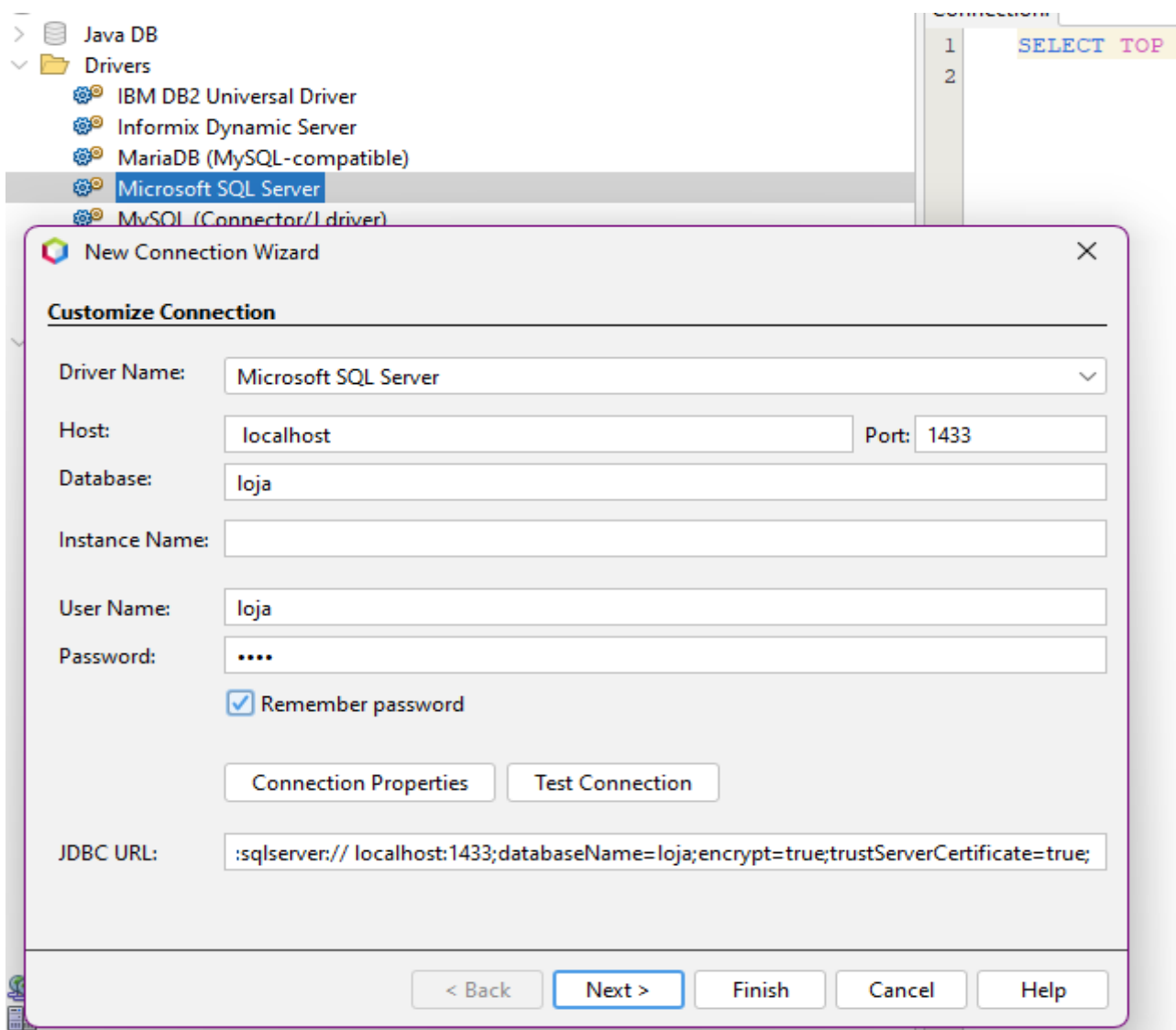
- O reconhecimento será automático, e podemos definir uma conexão com o clique do botão direito sobre o driver e escolha de Conectar Utilizando.

- Para os campos database, user e password, utilizar o valor loja, de acordo com os elementos criados em exercício anterior sobre a criação do banco de dados de exemplo, marcando também a opção Lembrar Senha.

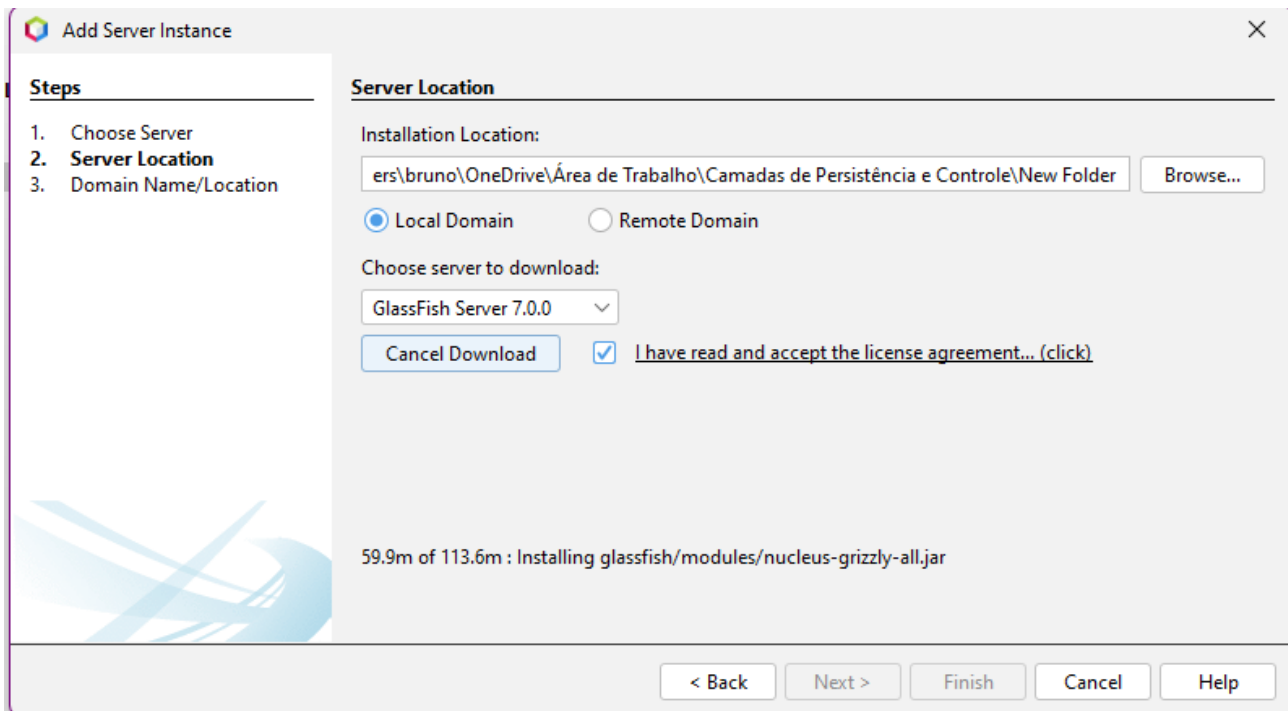
- Para o campo JDBC URL deve ser utilizada a seguinte expressão:

```
jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true
```

- Clicar em Testar Conexão e, estando tudo certo, Finalizar.

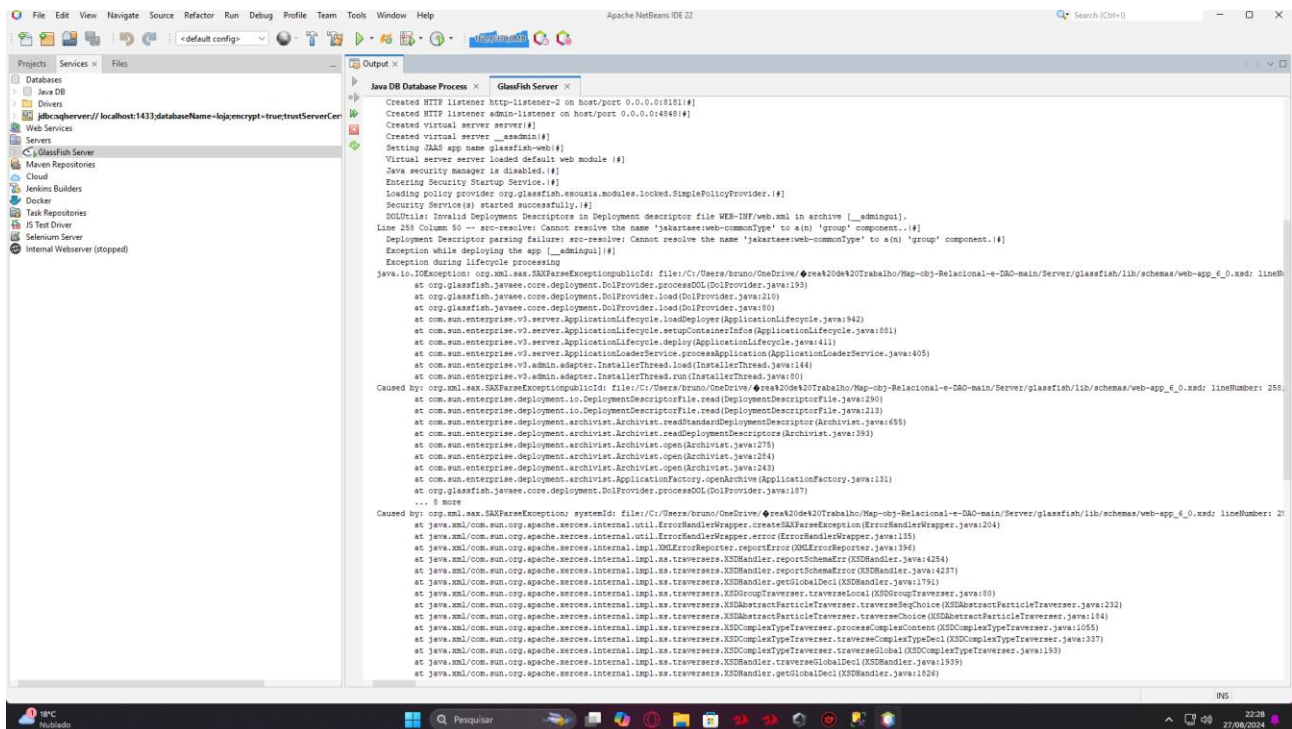


- Na divisão Servidores, verificar se o GlassFish 7.0.0 está instalado, e caso não esteja, adicionar o servidor, via clique com o botão direito e escolha da opção Add Server, efetuando o download a partir da própria janela que se abrirá



Adicionando o Servidor GlassFish 7.0.0, Obs.: Foram feitas algumas mudanças nos programas que foram solicitados no Trabalho como foi passado a versão que pede no trabalho e a 6.2.1 porém foi utilizado versão mais recentes

- Copiar o arquivo mssql-jdbc-12.2.0.jre8.jar para o subdiretório lib, a partir do diretório de base do GlassFish.
- Iniciar o servidor GlassFish a partir do NetBeans.



- Através da linha de comando, executar o comando asadmin, no diretório bin do GlassFish

- No prompt do asadmin, executar o comando apresentado a seguir:

```
create-jdbc-connection-pool --datasourceclassname com.microsoft.sqlserver.jdbc.SQLServerDataSource --restype javax.sql.DataSource --property driverClass=com.microsoft.sqlserver.jdbc.SQLServerDriver:portNumber=1433:password=loja:user=loja:serverName=localhost:databaseName=loja:trustServerCertificate=true:URL="jdbc\\:sqlserver\\://localhost\\:1433\\;databaseName\\=loja\\;encrypt\\=true\\;trustServerCertificate\\=true\\;"
```

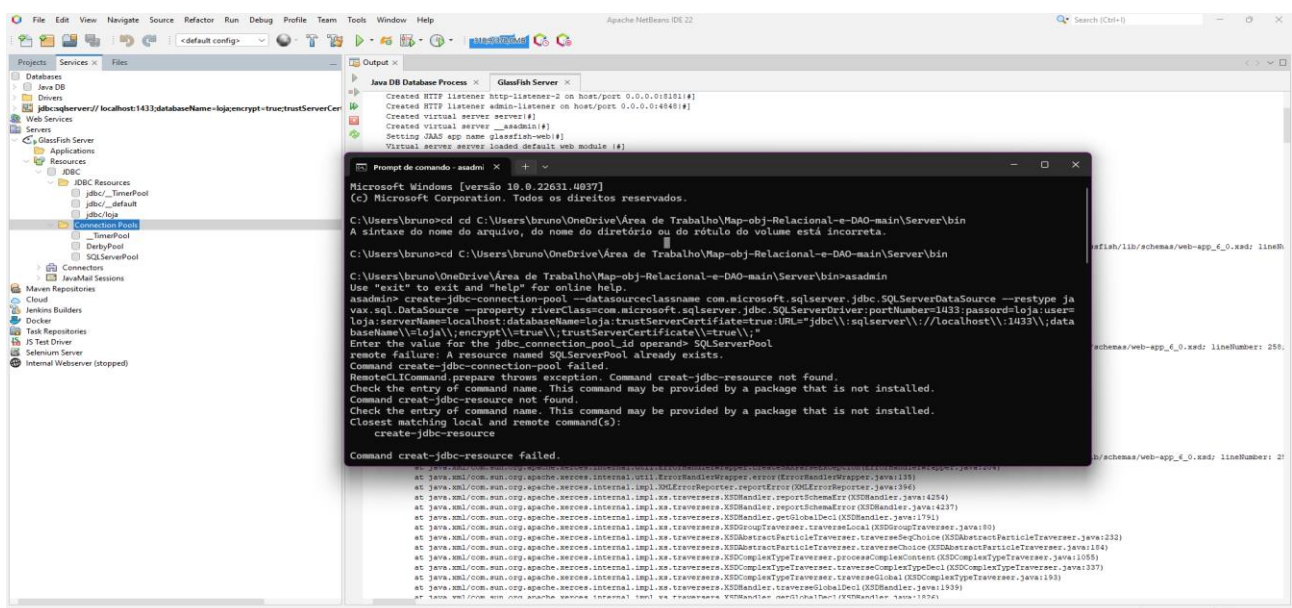
- Será solicitado o identificador do pool, que será SQLServerPool

- Testar o pool de conexões através do comando apresentado a seguir:

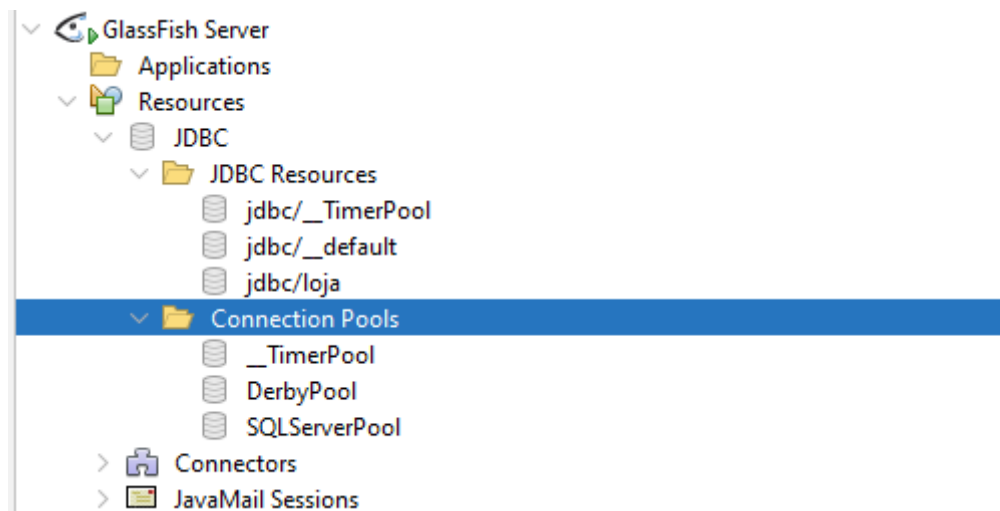
```
ping-connection-pool SQLServerPool
```

- Obtendo sucesso na operação, criar o registro JNDI, ainda no asadmin, através do comando apresentado a seguir:

```
create-jdbc-resource --connectionpoolid SQLServerPool jdbc/loja
```

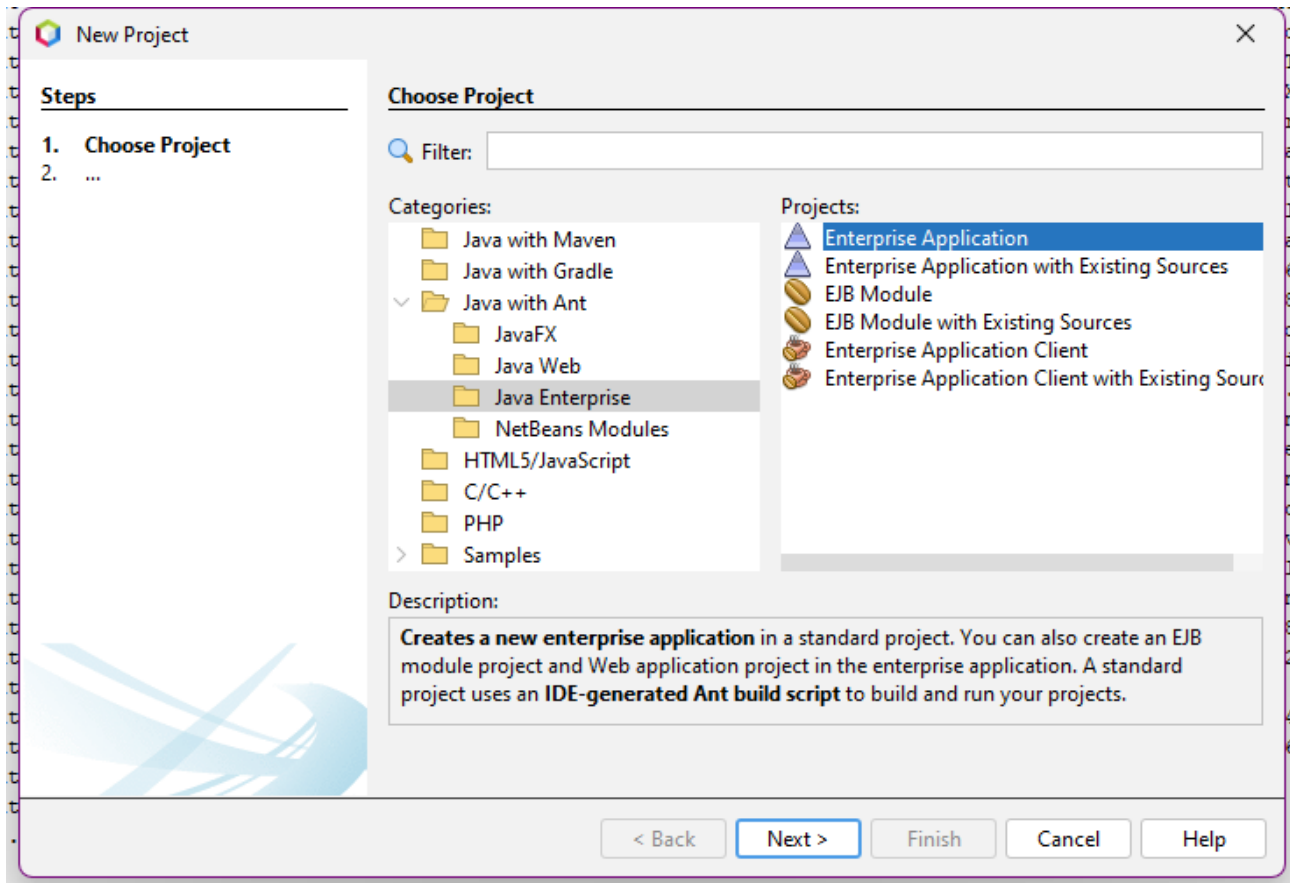


- Atualizar o servidor no ambiente do NetBeans e verificar se tudo foi gerado corretamente

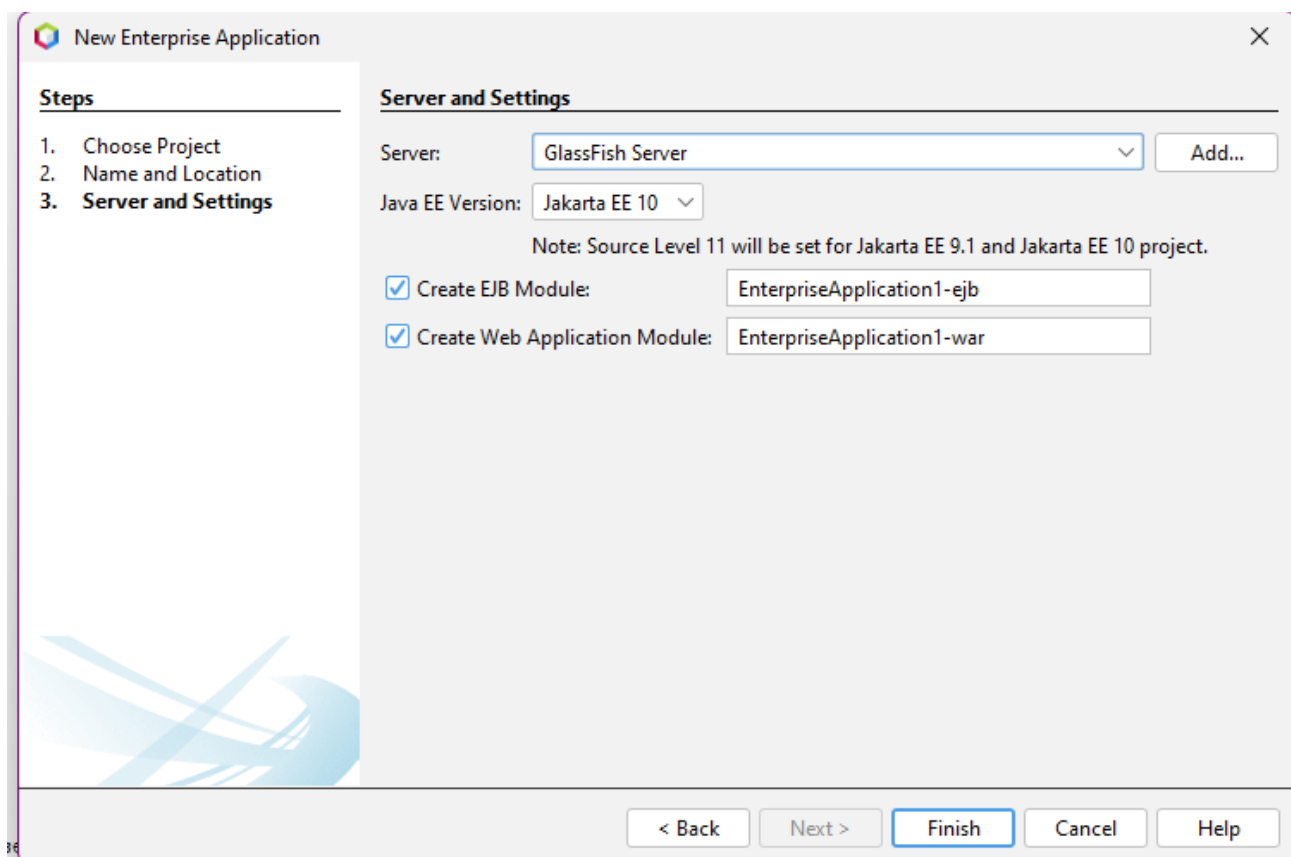


3 CRIAR O APLICATIVO CORPORATIVO NO NETBEANS:




- Criar um projeto do tipo Ant..Java Enterprise..Enterprise Application.



- Adotar o nome CadastroEE, com escolha do servidor GlassFish, além de plataforma Jakarta JEE 8.
- Serão gerados três projetos, onde o principal encapsula o arquivo EAR, tendo os outros dois, CadastroEE-ejb e CadastroEE-war, como projetos dependentes, relacionados aos elementos JPA, JEE e Web.

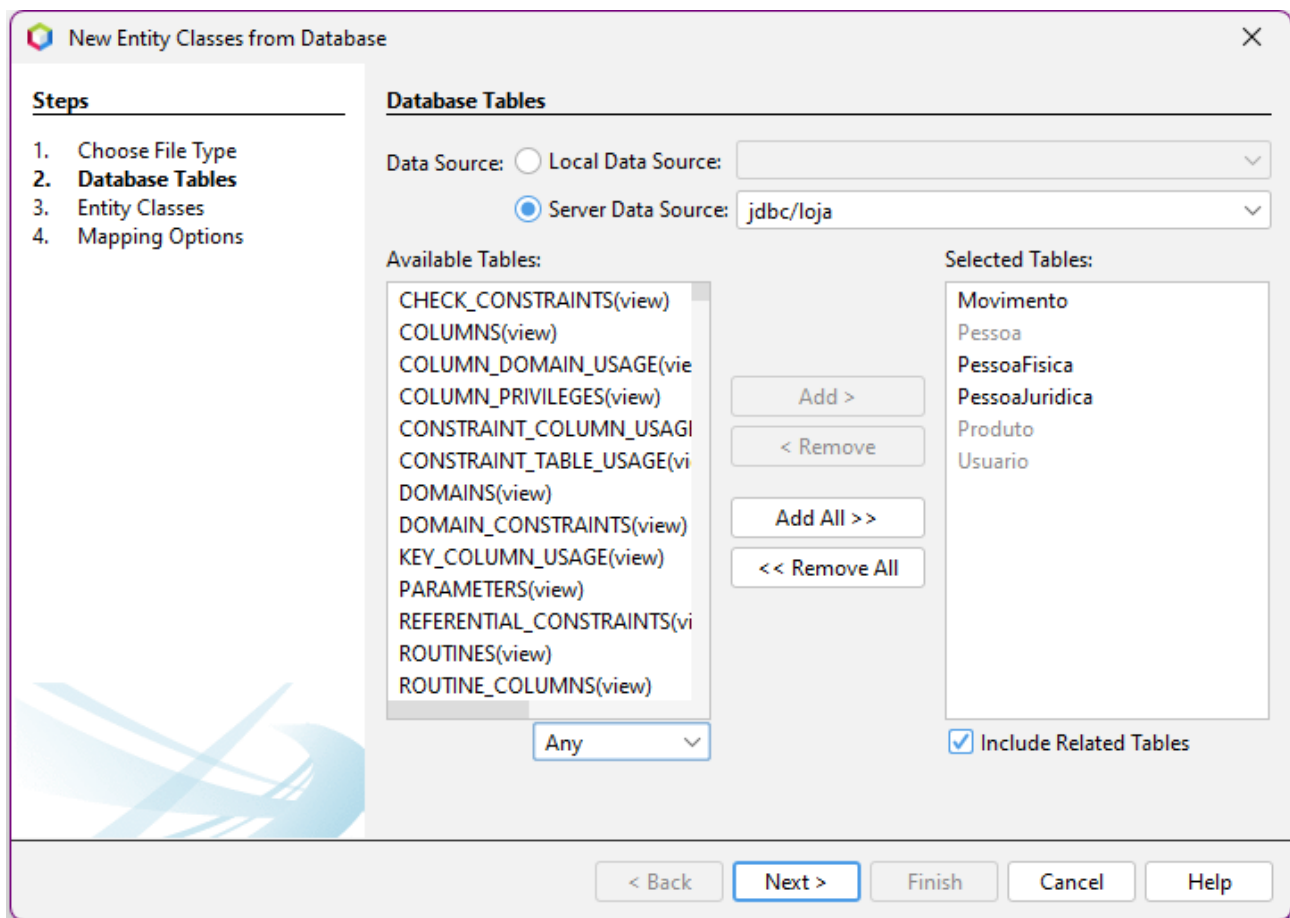


Adicionando o Jakarta EE 10, Obs.: Foram feitas algumas mudanças nos programas que foram solicitados no Trabalho como foi passado a versão que pede no trabalho Jakarta EE 8 porém foi utilizada versão mais recentes

- >  CadastroEE
- >  CadastroEE-ejb
- >  CadastroEE-war

4 DEFINIR AS CAMADAS DE PERSISTÊNCIA E CONTROLE NO PROJETO CADASTROEE-EJB.

- Criar as entidades JPA através de New Entity Classes from Database.
- Selecionar jdbc/loja como Data Source, e selecionar todas as tabelas.



- No passo seguinte, definir o pacote como cadastroee.model, além de marcar a opção para criação do arquivo persistence.xml.

New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
3. **Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
PessoaJuridica	PessoaJuridica	New
Produto	Produto	New
Usuario	Usuario	New

Project: CadastroEE-ejb

Location: Source Packages

Package: cadastroee.model

☒ Generate Named Query Annotations for Persistent Fields

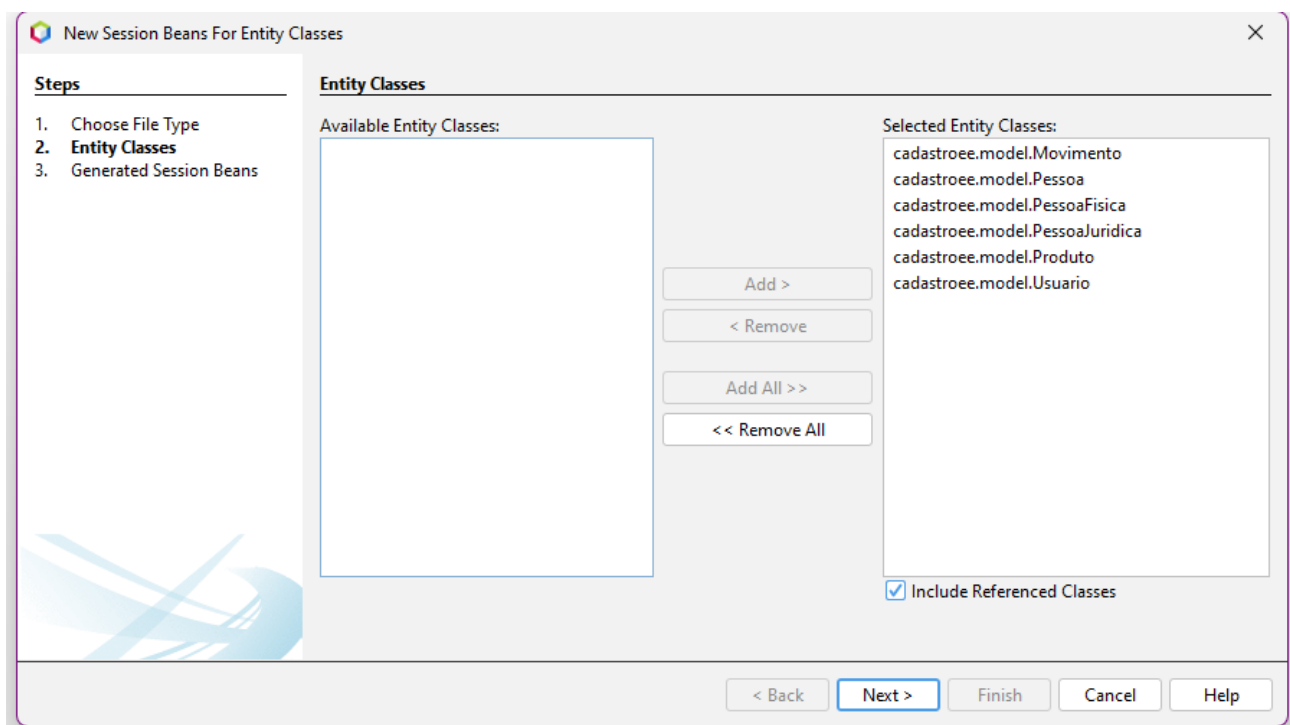
☒ Generate JAXB Annotations






























☐ Generate MappedSuperclasses instead of Entities

☒ Create Persistence Unit

< Back Next > Finish Cancel Help

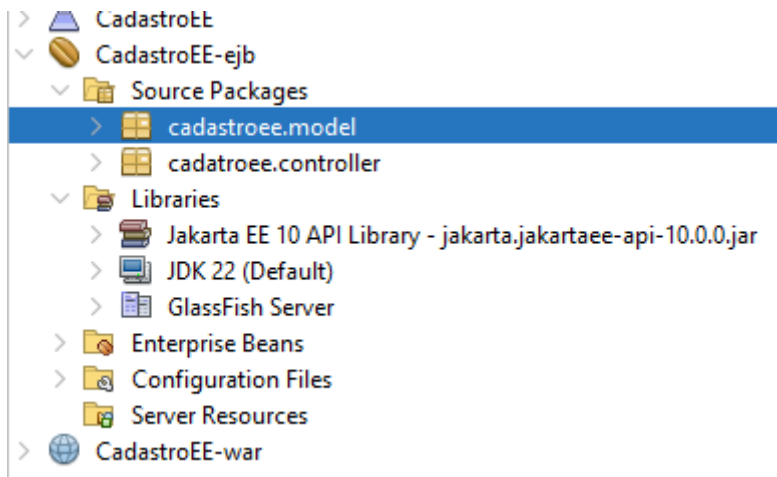
- Em seguida, adicionar os componentes EJB ao projeto, através da opção New Session Beans for Entity Classes.
- Selecionar todas as entidades, marcar a geração da interface local, além de definir o nome do pacote como cadastroee.controller.
- Serão gerados todos os Session Beans, com o sufixo Facade, bem como as interfaces, com o sufixo FacadeLocal.



- >  CadastroEE
- ✓  CadastroEE-ejb
 - ✓  Source Packages
 - ✓  cadastroee.model
 -  Movimento.java
 -  Pessoa.java
 -  PessoaFisica.java
 -  PessoaJuridica.java
 -  Produto.java
 -  Usuario.java
 - ✓  cadatroee.controller
 -  AbstractFacade.java
 -  MovimentoFacade.java
 -  MovimentoFacadeLocal.java
 -  PessoaFacade.java
 -  PessoaFacadeLocal.java
 -  PessoaFisicaFacade.java
 -  PessoaFisicaFacadeLocal.java
 -  PessoaJuridicaFacade.java
 -  PessoaJuridicaFacadeLocal.java
 -  ProdutoFacade.java
 -  ProdutoFacadeLocal.java
 -  UsuarioFacade.java
 -  UsuarioFacadeLocal.java
- >  Libraries
- >  Enterprise Beans
- >  Configuration Files
- >  Server Resources
- >  CadastroEE-war

5 EFETUAR PEQUENOS ACERTOS NO PROJETO, PARA USO DO JAKARTA:

- Adicionar a biblioteca Jakarta EE 10 API ao projeto CadastroEE-ejb.
- Criados os componentes e ajustadas as bibliotecas, o projeto deverá ficar como apresentado a seguir.



- Modificar TODAS as importações de pacotes javax para jakarta, em todos os arquivos do projeto CadastroEE-ejb.

```
package cadastroee.model;

import jakarta.persistence.Basic;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
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 jakarta.xml.bind.annotation.XmlTransient;
import java.io.Serializable;
import java.util.Collection;
```


- Na entidade Produto, mudar o tipo do atributo precoVenda para Float no lugar de BigDecimal.

```
public void setPrecoVenda(Float precoVenda) {  
    this.precoVenda = precoVenda;  
}
```

- Modificar o arquivo persistence.xml para o que é apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/
xml/ns/persistence/persistence\_1\_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/>
  </persistence-unit>
</persistence>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/
xml/ns/persistence/persistence_1_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/>
  </persistence-unit>
</persistence>
```

6 CRIAR UM SERVLET DE TESTE NO PROJETO CADASTROEE-WAR

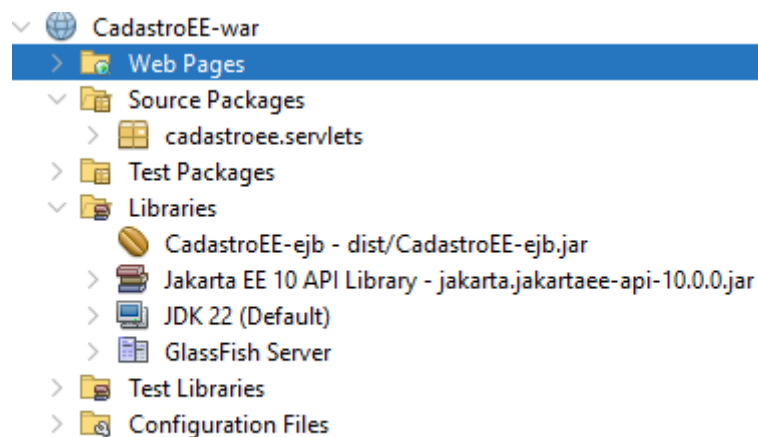
- Utilizar o clique do botão direito e escolha da opção New..Servlet
- Definir o nome do Servlet como ServletProduto, e nome do pacote como cadastroee.servlets
- Marcar opção Add information to deployment descriptor, algo que ainda é necessário quando o GlassFish 6 é utilizado
- Adicionar, no código do Servlet, a referência para a interface do EJB

@EJB ProdutoFacadeLocal facade

- Modificar a resposta do Servlet, utilizando o facade para recuperar os dados e apresentá-los na forma de lista HTML

7 EFETUAR NOVOS ACERTOS NO PROJETO, PARA USO DO JAKARTA:

- Adicionar a biblioteca Jakarta EE Web 10 API ao projeto CadastroEE-war
- Criado o Servlet e ajustadas as bibliotecas, o projeto deverá ficar como apresentado a seguir:



- Modificar TODAS as importações de pacotes javax para jakarta, em todos os arquivos do projeto CadastroEE-war

Modificar o arquivo web.xml para o que é apresentado a seguir:

```
<?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>
```

```
<session-config>
```

```
<session-timeout>30</session-timeout>
```

```
</session-config>
```

```
</web-app>
```

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

8 EXECUTAR O PROJETO:

- A execução deve ser efetuar com o uso de Run ou Deploy no projeto principal (CadastroEE), simbolizado por um triângulo
- Acessar o endereço a seguir, para testar o Servlet `http://localhost:8080/CadastroEE-war/ServletProduto`
- Tendo alimentado a base via SQL Server Management Studio, ou pela aba de serviços do NetBeans, deve ser obtida uma saída como a seguinte:



9 ANÁLISE E CONCLUSÃO:

- ***Como é organizado um projeto corporativo no NetBeans?***

Estrutura do Projeto:

Projetos podem ser divididos em módulos.

Código-fonte em src/main/java, testes em src/test/java.

Dependências gerenciadas em "Libraries".

Ferramentas de Build:

Maven ou Ant para gestão de dependências e configuração do projeto.

Ambientes:

Perfis para diferentes ambientes (desenvolvimento, teste, produção).

Controle de Versão:

Integração com Git ou SVN.

Integração Contínua:

Configuração com Jenkins ou outras ferramentas de CI.

Servidores:

Configuração de servidores de aplicação (GlassFish, TomEE).

Documentação:

Geração de Javadoc e manutenção de README e CHANGELOG.

Testes Automatizados:

Integração com JUnit ou TestNG.

Colaboração:

Integração com sistemas de tarefas como Jira.

Configuração:

Uso de arquivos *.properties ou application.yml para parâmetros configuráveis.

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

JPA (Java Persistence API)

Persistência de Dados: JPA é uma especificação que facilita a gestão de persistência de dados em bancos de dados relacionais. Ela permite que os desenvolvedores mapeiem objetos Java para tabelas do banco de dados, utilizando anotações ou XML.

ORM (Object-Relational Mapping): Com JPA, as classes Java são mapeadas para tabelas do banco de dados, e os objetos dessas classes são mapeados para as linhas dessas tabelas. Isso elimina a necessidade de escrever código SQL manualmente.

Independência de Banco de Dados: JPA abstrai as operações de banco de dados, tornando o aplicativo menos dependente de um banco de dados específico. Isso facilita a troca de bancos de dados se necessário.

Gerenciamento de Transações: JPA também integra com o gerenciamento de transações, permitindo que operações em banco de dados sejam automaticamente gerenciadas no contexto de transações.

EJB (Enterprise JavaBeans)

Componentes Empresariais: EJB é uma especificação para criar componentes de negócios robustos e escaláveis, que podem ser reutilizados em diferentes partes da aplicação.

Gerenciamento de Ciclo de Vida: O contêiner EJB gerencia automaticamente o ciclo de vida dos componentes EJB, lidando com aspectos como criação, destruição, passivação e ativação de beans.

Segurança e Transações: EJB oferece suporte integrado para segurança (autenticação e autorização) e gerenciamento de transações distribuídas, essencial para aplicações corporativas que necessitam garantir consistência e segurança.

Facilidade para Serviços Remotos: EJB permite que os componentes sejam acessados remotamente, facilitando a criação de serviços distribuídos e escaláveis, como microservices.

Tipos de EJBs:

Existem três tipos principais de EJBs:

Session Beans (Stateless e Stateful):

Usados para encapsular a lógica de negócios.

Message-Driven Beans:

Utilizados para processar mensagens de forma assíncrona.

Entity Beans:

Embora obsoletos em favor do JPA, esses eram utilizados para persistência de dados.

Como JPA e EJB se Integram:

EJB com JPA:

EJBs podem usar JPA para gerenciar a persistência de dados. Por exemplo, um Session Bean pode usar o EntityManager (parte da JPA) para realizar operações CRUD (Create, Read, Update, Delete) em entidades mapeadas.

Transações Gerenciadas:

O EJB pode gerenciar automaticamente as transações quando interage com JPA, garantindo que as operações de banco de dados sejam feitas dentro de um contexto transacional, o que é crucial para garantir a integridade dos dados.

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

Assistentes de Criação de Projetos:

Templates específicos configuram automaticamente o ambiente para JPA e EJB.

Geração Automática de Código:

Criação automática de entidades JPA e EJBs a partir de bancos de dados e templates.

Editor de Código Avançado:

Autocompletar, navegação rápida e IntelliSense aceleram o desenvolvimento.

Mapeamento Visual:

Ferramentas gráficas para visualizar mapeamentos JPA e diagramas de EJB.

Integração com Servidores:

Implantação e testes diretos em servidores de aplicação.

Ferramentas de Persistência:

Editor para JPQL e SQL, além de gerenciamento integrado de bancos de dados.

Testes Automatizados:

Facilita a criação e execução de testes para EJBs e operações JPA.

Depuração e Monitoração:

Depurador integrado e monitoração de transações ajudam na resolução de problemas.

- ***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 funcionam no lado do servidor para lidar com solicitações de clientes e gerar respostas dinâmicas. Eles são uma parte fundamental da tecnologia Java para desenvolvimento web, sendo frequentemente usados para processar requisições HTTP (como GET e POST), manipular dados enviados por formulários, manter sessões de usuário, e gerar conteúdo dinâmico, como páginas HTML.

Principais Características dos Servlets:

Interação com o Cliente:

Recebem e processam requisições de clientes (geralmente navegadores) e retornam respostas (como HTML, JSON, XML).

Ciclo de Vida:

São gerenciados pelo contêiner de servlets (como Apache Tomcat), que cuida da criação, inicialização, destruição e reciclagem de instâncias de servlets.

Escalabilidade e Reutilização:

Servlets são multithreaded, permitindo o atendimento de múltiplas solicitações simultaneamente com uma única instância.

- ***Como o NetBeans Oferece Suporte à Construção de Servlets?***

NetBeans oferece uma série de recursos que facilitam o desenvolvimento de servlets em projetos web:

Assistente de Criação de Servlets:

Templates de Servlet:

NetBeans possui assistentes que geram automaticamente o código base de um servlet, incluindo métodos essenciais como doGet() e doPost().

Configuração Automática:

O assistente também configura o arquivo web.xml ou adiciona anotações diretamente na classe, registrando o servlet e mapeando URLs.

Editor de Código Avançado:

Autocompletar e IntelliSense:

Durante a codificação, NetBeans oferece autocompletar para métodos e atributos, sugerindo rapidamente opções relevantes, como métodos HTTP, cabeçalhos e parâmetros de requisição.

Validação de Código:

O editor identifica erros de sintaxe e problemas de configuração em tempo real, facilitando a correção antes da execução.

Depuração e Testes:

Depuração Integrada:

NetBeans permite a depuração de servlets diretamente, possibilitando a inspeção de variáveis, pontos de interrupção e análise de fluxo de execução durante uma requisição.

Execução em Servidores Integrados:

A IDE pode executar e testar servlets em servidores de aplicação integrados, como Apache Tomcat, sem precisar sair da interface.

Integração com Servidores de Aplicação:

Implantação Simplificada:

Servidores como Apache Tomcat podem ser configurados e gerenciados diretamente no NetBeans, permitindo a implantação e execução contínua dos servlets durante o desenvolvimento.

Hot Deploy:

Alterações nos servlets podem ser automaticamente refletidas no servidor sem necessidade de recompilar e reiniciar todo o aplicativo.

Ferramentas para Web e Interface:

Suporte a JSP:

NetBeans oferece integração com JSP (JavaServer Pages), permitindo que servlets e páginas JSP trabalhem juntos na geração de conteúdo dinâmico.

Editor de HTML/CSS/JavaScript:

Para projetos web, o NetBeans inclui editores avançados para HTML, CSS e JavaScript, integrando o frontend com os servlets.

Integração com Bibliotecas e Frameworks:

Suporte a Frameworks como Spring:

NetBeans facilita a integração de servlets com frameworks populares, como Spring MVC, permitindo a criação de controladores baseados em servlets de forma mais estruturada.

Ferramentas de Monitoramento e Desempenho:

Monitoramento de Requisições:

NetBeans oferece ferramentas para monitorar o desempenho e o comportamento dos servlets durante a execução, ajudando na identificação de gargalos.

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

Injeção via @EJB:

A forma mais comum de comunicação é através da anotação `@EJB`, que injeta automaticamente a instância do Session Bean no Servlet.

Lookup JNDI:

Alternativamente, o Servlet pode realizar um lookup manual usando JNDI para obter uma referência ao EJB, útil em cenários mais complexos.

Chamadas Remotas:

Se o EJB estiver em uma JVM diferente, o Servlet pode acessar o EJB via interface remota usando JNDI.

Gerenciamento de Transações:

As transações são gerenciadas automaticamente pelo contêiner, garantindo consistência e segurança nas operações entre Servlets e EJBs.

10 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 centraliza a gestão de requisições em uma aplicação web. Em uma arquitetura MVC, ele funciona como um único Servlet que intercepta todas as requisições, determina qual controlador específico deve processá-las, e encaminha a resposta para a View apropriada.

Front Controller Servlet:

Intercepta todas as requisições e decide qual ação (Controller) deve ser executada.

Despacho para Controladores:

Com base na URL ou parâmetros, o Front Controller encaminha a requisição para um controlador específico que executa a lógica de negócios.

Seleção da View:

Após o processamento, o Front Controller escolhe e exibe a View (como uma página JSP) para o usuário.

Essa abordagem centraliza o fluxo de controle, facilitando a manutenção e a implementação de funcionalidades transversais como autenticação e log.

- ***Quais as diferenças e semelhanças entre Servlets e JSPs?***

Semelhanças entre Servlets e JSPs

Tecnologia Java EE:

Ambos são usados para criar aplicações web dinâmicas em Java.

Geração de Conteúdo Dinâmico:

Ambos podem gerar conteúdo dinâmico baseado em requisições HTTP.

Integração:

Servlets e JSPs podem trabalhar juntos na mesma aplicação.

Diferenças entre Servlets e JSPs

Modelo de Programação:

Servlets:

Classes Java que processam requisições e geram respostas.

JSPs:

Arquivos HTML com código Java embutido, focados na apresentação.

Linguagem e Sintaxe:

Servlets:

Código Java puro, mais estruturado e verboso.

JSPs:

HTML misturado com Java, mais amigável para criar páginas.

Geração e Compilação:

Servlets:

Compilados diretamente como classes Java.

JSPs:

Convertidos em Servlets e compilados pelo contêiner de servlets.

Responsabilidade:***Servlets:***

Gerenciam a lógica de controle e processamento.

JSPs:

Focam na apresentação e formatação de dados.

Manutenção e Clareza:***Servlets:***

Misturam lógica de negócios e apresentação, podendo ser mais difíceis de manter.

JSPs:

Separação clara entre lógica de negócios e apresentação, facilitando a manutenção.

- ***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?***

Diferença entre Redirecionamento e Forward

Redirecionamento (Redirect):

Método:

`response.sendRedirect("url")`

Funcionamento:

O redirecionamento faz com que o navegador do cliente envie uma nova solicitação para a URL especificada. Isso significa que o cliente vê a mudança de URL e uma nova requisição HTTP é criada.

Efeito:

O redirecionamento resulta em uma nova URL no navegador e uma nova requisição ao servidor. Qualquer dado da requisição original não é enviado para o novo destino.

Forward (Encaminhamento):

Método:

```
request.getRequestDispatcher("url").forward(request, response)
```

Funcionamento:

O encaminhamento é feito no lado do servidor. O servidor processa a requisição e a encaminha para outra página ou servlet sem que o cliente perceba a mudança. O navegador mantém a mesma URL.

Efeito:

O encaminhamento mantém a URL original e transfere o controle para o novo recurso (como um JSP ou outro servlet) enquanto preserva os dados da requisição original.

Parâmetros e Atributos no HttpServletRequest

Parâmetros:

Definição:

Parâmetros são dados enviados pelo cliente para o servidor, geralmente via query string em URLs ou via corpo de solicitações POST.

Uso:

São usados para passar dados simples, como valores de formulários ou parâmetros de URL.

Atributos:**Definição:**

Atributos são dados que o servidor pode adicionar à requisição durante o processamento e que podem ser compartilhados entre servlets e JSPs.

Uso:

São usados para passar dados complexos, como objetos ou informações que precisam ser acessadas por várias partes da aplicação durante o ciclo de vida da requisição.

Resumo**Redirecionamento:**

Cria uma nova requisição e URL visível para o cliente. Não mantém dados da requisição original.

Forward:

Processa a requisição no servidor sem mudar a URL visível para o cliente e mantém os dados da requisição original.

Parâmetros:

Dados enviados pelo cliente para o servidor, acessíveis via `request.getParameter()`.

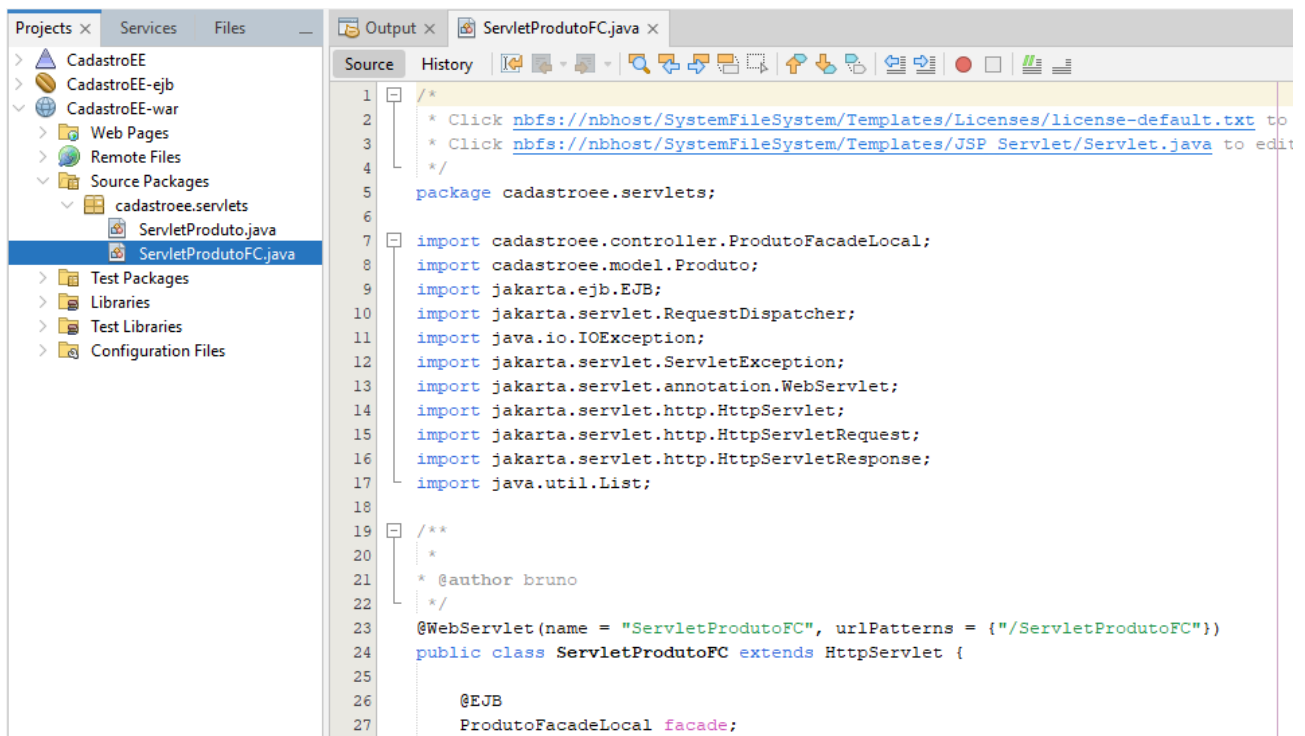
Atributos:

Dados adicionados pelo servidor à requisição, acessíveis via `request.getAttribute()` e compartilháveis entre servlets e JSPs.

11 2º PROCEDIMENTO | INTERFACE CADASTRAL COM SERVLET E JSPS

12 CRIAR UM SERVLET COM O NOME SERVLETPRODUTOFC, NO PROJETO CADASTROEE-WAR:

- Utilizar o padrão Front Controller
- Adicionar uma referência para ProdutoFacadeLocal, utilizando o nome facade para o atributo



The screenshot shows an IDE with the 'Projects' tab selected. The project structure on the left includes 'CadastroEE', 'CadastroEE-ejb', and 'CadastroEE-war'. Under 'CadastroEE-war', there are 'Web Pages', 'Remote Files', and 'Source Packages'. The 'Source Packages' folder contains 'cadastroee.servlets', which includes 'ServletProduto.java' and 'ServletProdutoFC.java'. The 'ServletProdutoFC.java' file is selected and its source code is displayed in the main editor. The code includes package declarations, imports for 'ProdutoFacadeLocal', 'Produto', and various Jakarta Servlet and EJB classes, a Javadoc comment, and the start of the 'ServletProdutoFC' class extending 'HttpServlet'.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3  * Click nbfs://nbhost/SystemFileSystem/Templates/JSP_Servlet/Servlet.java to edit
4  */
5  package cadastroee.servlets;
6
7  import cadastroee.controller.ProdutoFacadeLocal;
8  import cadastroee.model.Produto;
9  import jakarta.ejb.EJB;
10 import jakarta.servlet.RequestDispatcher;
11 import java.io.IOException;
12 import jakarta.servlet.ServletException;
13 import jakarta.servlet.annotation.WebServlet;
14 import jakarta.servlet.http.HttpServlet;
15 import jakarta.servlet.http.HttpServletRequest;
16 import jakarta.servlet.http.HttpServletResponse;
17 import java.util.List;
18
19 /**
20 *
21 * @author bruno
22 */
23 @WebServlet(name = "ServletProdutoFC", urlPatterns = {"/ServletProdutoFC"})
24 public class ServletProdutoFC extends HttpServlet {
25
26     @EJB
27     ProdutoFacadeLocal facade;
```

- Apagar o conteúdo interno do método processRequest, e efetuar nele as modificações seguintes
- Capturar o parâmetro acao a partir do request, o qual poderá assumir os valores listar, incluir, alterar, excluir, formIncluir e formAlterar
- Definir a variável destino, contendo o nome do JSP de apresentação, que terá os valores ProdutoDados.jsp, para acao valendo formAlterar ou formIncluir, ou ProdutoLista.jsp, para as demais opções
- Para o valor listar, adicionar a listagem de produtos como atributo da requisição (request), com a consulta efetuada via facade
- Para o valor formAlterar, capturar o id fornecido como parâmetro do request, consultar a entidade via facade, e adicioná-la como atributo da requisição (request)
- Para o valor excluir, capturar o id fornecido como parâmetro do request, remover a entidade através do facade, e adicionar a listagem de produtos como atributo da requisição (request)
- Para o valor alterar, capturar o id fornecido como parâmetro do request, consultar a entidade através do facade, preencher os demais campos com os valores fornecidos no request, alterar os dados via facade e adicionar a listagem de produtos como atributo da requisição (request)
- Para o valor incluir, instanciar uma entidade do tipo Produto, preencher os campos com os valores fornecidos no request, inserir via facade e adicionar a listagem de produtos como atributo da requisição (request)
- Ao final redirecionar para destino via RequestDispatcher, obtido a partir do objeto request

```
Output x ServletProdutoFC.java x
Source History
1 1
2 2
3 3
4 4
5 package cadastroee.servlets;
6
7 import cadastroee.controller.ProdutoFacadeLocal;
8 import cadastroee.model.Produto;
9 import jakarta.ejb.EJB;
10 import jakarta.servlet.RequestDispatcher;
11 import java.io.IOException;
12 import jakarta.servlet.ServletException;
13 import jakarta.servlet.annotation.WebServlet;
14 import jakarta.servlet.http.HttpServlet;
15 import jakarta.servlet.http.HttpServletRequest;
16 import jakarta.servlet.http.HttpServletResponse;
17 import java.util.List;
18
19 /**
20 *
21 * @author bruno
22 */
23 @WebServlet(name = "ServletProdutoFC", urlPatterns = {"/ServletProdutoFC"})
24 public class ServletProdutoFC extends HttpServlet {
25
26     @EJB
27     ProdutoFacadeLocal facade;
28
29     @Override
30     protected void doGet(HttpServletRequest request, HttpServletResponse response)
31         throws ServletException, IOException {
32
33         String acao = request.getParameter("acao");
34         String destino = "";
35
36         switch (acao) {
37             case "formIncluir":
38                 destino = "ProdutoDados.jsp";
39                 break;
40
41             case "excluir":
42                 int idDel = Integer.parseInt(request.getParameter("id"));
43                 facade.remove(facade.find(idDel));
44                 List<Produto> delProdutos = facade.findAll();
45                 request.setAttribute("produtos", delProdutos);
46                 destino = "ProdutoLista.jsp";
47                 break;
48 }
```

```
Output x ServletProdutoFC.java x
Source History
46         destino = "ProdutoLista.jsp";
47         break;
48
49         case "formAlterar":
50             int id = Integer.parseInt(request.getParameter("id"));
51             Produto produto = facade.find(id);
52             request.setAttribute("produto", produto);
53             destino = "ProdutoDados.jsp";
54             break;
55
56         default:
57             List<Produto> produtos = facade.findAll();
58             request.setAttribute("produtos", produtos);
59             destino = "ProdutoLista.jsp";
60             break;
61     }
62
63     RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
64     dispatcher.forward(request, response);
65
66 }
67
68 @Override
69 protected void doPost(HttpServletRequest request, HttpServletResponse response)
70     throws ServletException, IOException {
71
72     String acao = request.getParameter("acao");
73     acao = acao == null || acao.isEmpty() ? " " : acao;
74
75     String destino = "ProdutoLista.jsp";
76
77     switch (acao) {
78
79         case "incluir":
80             int idProduto = Integer.parseInt(request.getParameter("idProduto"));
81             String nome = request.getParameter("nome");
82             int quantidade = Integer.parseInt(request.getParameter("quantidade"));
83             Float precoVenda = Float.valueOf(request.getParameter("precoVenda"));
84
85             Produto newProduto = new Produto();
86             newProduto.setIdProduto(idProduto);
87             newProduto.setNome(nome);
88             newProduto.setQuantidade(quantidade);
89             newProduto.setPrecoVenda(precoVenda);
90
91             facade.create(newProduto);
92
93             List<Produto> newProdutos = facade.findAll();
94
95     cadastroee.servlets.ServletProdutoFC > doGet > switch (acao) > case "excluir": >
```

```
Output x ServletProdutoFC.java x
Source History
88     newProduto.setQuantidade(quantidade);
89     newProduto.setPrecoVenda(precoVenda);
90
91     facade.create(newProduto);
92
93     List<Produto> newProdutos = facade.findAll();
94     request.setAttribute("produtos", newProdutos);
95     break;
96
97     case "alterar":
98         Produto alterarProduto = facade.find(Integer.valueOf(request.getParameter("id")));
99
100         String alterarNome = request.getParameter("nome");
101         int alterarQuantidade = Integer.parseInt(request.getParameter("quantidade"));
102         Float alterarPrecoVenda = Float.valueOf(request.getParameter("precoVenda"));
103
104         alterarProduto.setNome(alterarNome);
105         alterarProduto.setQuantidade(alterarQuantidade);
106         alterarProduto.setPrecoVenda(alterarPrecoVenda);
107
108         facade.edit(alterarProduto);
109         List<Produto> alterarProdutos = facade.findAll();
110         request.setAttribute("produtos", alterarProdutos);
111         break;
112
113     default:
114         List<Produto> produtos = facade.findAll();
115         request.setAttribute("produtos", produtos);
116         break;
117 }
118
119 RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
120 dispatcher.forward(request, response);
121 }
122
123 }
124
```

13 CRIAR A PÁGINA DE CONSULTA, COM O NOME PRODUTOLISTA.JSP

- Incluir um link para ServletProdutoFC, com acao formIncluir, voltado para a abertura do formulário de inclusão.
- Definir uma tabela para apresentação dos dados.
- Recuperar a lista de produtos enviada pelo Servlet.
- Para cada elemento da lista, apresentar id, nome, quantidade e preço como células da tabela.
- Criar, também, de forma dinâmica, links para alteração e exclusão, com a chamada para ServletProdutoFC, passando as ações corretas e o id do elemento corrente.
- Organizar o código para obter uma página como a seguinte.

Listagem de Produtos

ID	Produto	Quantidade	Preço	Ações	
1	Banana	100	R\$ 5,00	Alterar	Excluir

14 CRIAR A PÁGINA DE CADASTRO, COM O NOME PRODUTODADOS.JSP

- Definir um formulário com envio para ServletProdutoFC, modo post.
- Recuperar a entidade enviada pelo Servlet.
- Definir a variável acao, com valor incluir, para entidade nula, ou alterar, quando a entidade é fornecida.
- Incluir um campo do tipo hidden, para envio do valor de acao .
- Incluir um campo do tipo hidden, para envio do id, apenas quando o valor de acao for alterar.
- Incluir os campos para nome, quantidade e preço de venda, preenchendo os dados quando a entidade é fornecida.
- Concluir o formulário com um botão de envio, com o texto adequado para as situações de inclusão ou alteração de dados .
- Organizar o código para obter uma página como a seguinte.

Cadastro de Produto

[Voltar](#)

Nome

Banana


Quantidade

100

Preço de Venda

4,00

[Cadastrar](#)


```
Source History 
1 1 /*
2 2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3 3  * Click nbfs://nbhost/SystemFileSystem/Templates/JSP\_Servlet/Servlet.java to edit this template
4 4  */
5 5 package cadastroee.servlets;
6 6
7 7 import cadastroee.controller.ProdutoFacadeLocal;
8 8 import cadastroee.model.Produto;
9 9 import jakarta.ejb.EJB;
10 10 import jakarta.servlet.RequestDispatcher;
11 11 import java.io.IOException;
12 12 import jakarta.servlet.ServletException;
13 13 import jakarta.servlet.annotation.WebServlet;
14 14 import jakarta.servlet.http.HttpServlet;
15 15 import jakarta.servlet.http.HttpServletRequest;
16 16 import jakarta.servlet.http.HttpServletResponse;
17 17 import java.util.List;
18 18
19 19 /**
20 20  *
21 21  * @author bruno
22 22  */
23 23 @WebServlet(name = "ServletProdutoFC", urlPatterns = {"/ServletProdutoFC"})
24 24 public class ServletProdutoFC extends HttpServlet {
25 25
26 26     @EJB
27 27     ProdutoFacadeLocal facade;
28 28
29 29     @Override
30 30     protected void doGet(HttpServletRequest request, HttpServletResponse response)
31 31         throws ServletException, IOException {
32 32
33 33         String acao = request.getParameter("acao");
34 34         String destino = "";
35 35
36 36         switch (acao) {
37 37             case "formIncluir":
38 38                 destino = "ProdutoDados.jsp";
39 39                 break;
40 40
41 41             case "excluir":
42 42                 int idDel = Integer.parseInt(request.getParameter("id"));
43 43                 facade.remove(facade.find(idDel));
44 44                 List<Produto> delProdutos = facade.findAll();
45 45                 request.setAttribute("produtos", delProdutos);
46 46                 destino = "ProdutoLista.jsp";
47 47                 break;
48 48

```

```

        destino = "ProdutoLista.jsp";
        break;

        case "formAlterar":
            int id = Integer.parseInt(request.getParameter("id"));
            Produto produto = facade.find(id);
            request.setAttribute("produto", produto);
            destino = "ProdutoDados.jsp";
            break;

        default:
            List<Produto> produtos = facade.findAll();
            request.setAttribute("produtos", produtos);
            destino = "ProdutoLista.jsp";
            break;
    }

    RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
    dispatcher.forward(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String acao = request.getParameter("acao");
    acao = acao == null || acao.isEmpty() ? " " : acao;

    String destino = "ProdutoLista.jsp";

    switch (acao) {

        case "incluir":
            int idProduto = Integer.parseInt(request.getParameter("idProduto"));
            String nome = request.getParameter("nome");
            int quantidade = Integer.parseInt(request.getParameter("quantidade"));
            Float precoVenda = Float.valueOf(request.getParameter("precoVenda"));

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

            facade.create(newProduto);

            List<Produto> newProdutos = facade.findAll();

```

```

        newProduto.setQuantidade(quantidade);
        newProduto.setPrecoVenda(precoVenda);

        facade.create(newProduto);

        List<Produto> newProdutos = facade.findAll();
        request.setAttribute("produtos", newProdutos);
        break;

    case "alterar":
        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);
        List<Produto> alterarProdutos = facade.findAll();
        request.setAttribute("produtos", alterarProdutos);
        break;

    default:
        List<Produto> produtos = facade.findAll();
        request.setAttribute("produtos", produtos);
        break;
    }

    RequestDispatcher dispatcher = request.getRequestDispatcher(destino);
    dispatcher.forward(request, response);
}
}

```

15 TESTAR AS FUNCIONALIDADES DO SISTEMA:

- Listar os produtos com a chamada para o endereço seguinte:

<http://localhost:8080/CadastroEE-war/ServletProdutoFC?acao=listar>

- Efetuar uma inclusão a partir do link da tela de listagem
- Efetuar uma alteração a partir do link dinâmico da listagem
- Efetuar uma exclusão a partir do link dinâmico da listagem

ID	Produto	Quantidade	Preço	Ações
1	Banana	100	R\$ 5,00	Alterar Excluir
2	Laranja	500	R\$ 2,00	Alterar Excluir
				Cadastrar Produto

Inclusão

Cadastro de Produto

Voltar

Nome

Quantidade

Preço de Venda

Cadastrar

Alterar

Alteração de Produto

Voltar

Nome

Quantidade

Preço de Venda

Alterar

Listagem de Produtos

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

Exclusão

Listagem de Produtos

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

Cadastrar Produto

16 3º PROCEDIMENTO | MELHORANDO O DESIGN DA INTERFACE

- Incluir as bibliotecas do framework Bootstrap nos arquivos ProdutoLista.jsp e ProdutoDados.jsp
- Visitar o site do Bootstrap, no endereço <https://getbootstrap.com/>
- Rolar para baixo até encontrar a inclusão via CDN
- Clicar no botão para cópia do link CSS e colar na divisão head de cada uma das páginas JSP.
- Clicar no botão para cópia do link para a biblioteca Java Script e colar na divisão head de cada uma das páginas JSP

17 MODIFICAR AS CARACTERÍSTICAS DE PRODUTOLISTA.JSP

- Adicionar a classe container ao body.
- Adicionar as classes btn, btn-primary e m-2 no link de inclusão.
- Adicionar as classes table e table-striped na tabela.
- Adicionar a classe table-dark ao thead.
- Adicionar as classes btn, btn-primary e btn-sm no link de alteração.
- Adicionar as classes btn, btn-danger e btn-sm no link de exclusão.
- Ajustar as características para obter o design apresentado a seguir.

Listagem de Produtos

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

Cadastrar Produto

18 MODIFICAR AS CARACTERÍSTICAS DE PRODUTODADOS.JSP

- Adicionar a classe container ao body.
- Encapsule cada par label / input em div com classe mb-3.
- Adicionar a classe form ao formulário.
- Adicionar a classe form-label em cada label.
- Adicionar a classe form-control em cada input.
- Adicionar as classes btn e btn-primary ao botão de inclusão.
- Ajustar as características para obter o design apresentado a seguir.



A interface apresenta um formulário de alteração de produto. No topo, há uma barra azul com o título "Alteração de Produto". Abaixo, um botão cinza "Voltar" está no canto superior esquerdo. O formulário principal, com fundo branco e sombra, contém os seguintes campos:

- Nome:** Campo de texto com o valor "Tangerina".
- Quantidade:** Campo de texto com o valor "600".
- Preço de Venda:** Campo de texto com o valor "7.0".

No canto inferior direito do formulário, há um botão azul "Alterar".

Código do ProdutoLista.jsp

```
<%@ page import="java.text.DecimalFormat" %>
<%@ page import="cadastroee.model.Produto" %>
<%@ page import="java.util.List" %>
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<title>Listagem de Produtos</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.m
<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;
    }
```

```

        .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>

                    <tbody>
                        <tr>
                            <td colspan="5">
                                <!-- Decimals -->
                                <!-- Listagem de produtos -->
                                <!-- if -->
                                <!-- for -->
                                <!-- td -->
                                <!-- td -->
                                <!-- td -->
                                <!-- td -->
                                <!-- td -->
                                <!-- a -->
                                <!-- a -->
                            </td>
                        </tr>
                    </tbody>
                </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>

```

Codigo do ProdutoDados.jsp

```

<%page import="java.text.DecimalFormat"%>
<%page import="cadastroroe.model.Produto"%>
<%page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Cadastro de Produtos</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-4bw+...>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-Hwwvtg...>
<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;
    }
    .form-container {
        background-color: white;
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .form-label {
        font-weight: bold;
        color: #333;
    }
    .btn-primary {
        background-color: #007bff;
        border-color: #007bff;
    }
    .btn-secondary {
        margin-bottom: 15px;
    }
</style>
</head>
<body class="container">
<%
    DecimalFormat df = new DecimalFormat("#,##0.00");
</style>
</head>
<body class="container">
<%
    DecimalFormat df = new DecimalFormat("#,##0.00");
    Produto produto = (Produto) request.getAttribute("produto");
    String acao = produto != null ? "alterar" : "incluir";
%>
<div class="header-section text-center">
<h1><%=acao == "alterar" ? "Alteração" : "Cadastro"%> de Produto</h1>
</div>
<div class="row">
<div class="col-md-6 offset-md-3">
<div>
<a class="btn btn-secondary" href="ServletProdutoFC?acao=listar">Voltar</a>
</div>
<form class="form-container" action="ServletProdutoFC" method="post">
<input type="hidden" name="acao" value="<%=acao%>">
<% if (produto != null) { %>
<input type="hidden" name="id" value="<%=produto.getIdProduto()%>">
<% } %>
<div class="mb-3">
<label class="form-label" for="nome">Nome</label>
<input class="form-control" type="text" name="nome" value="<%=produto != null ? produto.getNome() : ""%>" required>
</div>
<div class="mb-3">
<label class="form-label" for="quantidade">Quantidade</label>
<input class="form-control" type="text" name="quantidade" value="<%=produto != null ? produto.getQuantidade() : ""%>" required>
</div>
<div class="mb-3">
<label class="form-label" for="precoVenda">Preço de Venda</label>
<input class="form-control" type="text" name="precoVenda" value="<%=produto != null ? produto.getPrecoVenda() : ""%>" required>
</div>
<div>
<input class="btn btn-primary" type="submit" value="<%=acao == "incluir" ? "Cadastrar" : "Alterar"%>">
</div>
</form>
</div>
</div>
</body>
</html>

```


19 ANÁLISE E CONCLUSÃO:

- ***Como o framework Bootstrap é utilizado?***

Inclusão:

CDN:

Adicione links para os arquivos CSS e JS diretamente no HTML.

Local:

Baixe e inclua os arquivos CSS e JS no seu projeto.

Estrutura:

Grid System:

Crie layouts responsivos usando um sistema de 12 colunas.

Componentes:

Utilize componentes pré-definidos como botões, formulários e barras de navegação.

Personalização:

Temas:

Modifique variáveis SCSS ou adicione CSS personalizado para ajustar o estilo.

Extensibilidade:

Plugins:

Use plugins JavaScript do Bootstrap para funcionalidades interativas como modais e tooltips.

Bootstrap facilita a criação de interfaces web responsivas e estilizadas com um conjunto de ferramentas e componentes prontos para uso.

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

Sistema de Grid Flexível

Grid Responsivo:

Bootstrap utiliza um sistema de grid baseado em 12 colunas, que se adapta a diferentes tamanhos de tela. Isso permite que você defina a estrutura de layout sem precisar modificar o HTML para diferentes resoluções.

Classes Utilitárias:

As classes CSS fornecidas pelo Bootstrap ajustam o layout de acordo com a resolução, garantindo que o HTML permaneça consistente e independente de ajustes específicos para dispositivos.

Componentes Reutilizáveis

Componentes Pré-Definidos:

Bootstrap oferece uma vasta gama de componentes pré-estilizados (como botões, formulários e barras de navegação) que você pode usar diretamente no HTML. Esses componentes têm um estilo consistente e são facilmente integráveis sem necessidade de customizações HTML.

Customização via Classes:

Em vez de modificar o HTML, você aplica classes do Bootstrap aos elementos HTML para alterar a aparência e o comportamento, mantendo o HTML limpo e independente de estilos específicos.

CSS e JavaScript Separados

Separação de Preocupações:

O Bootstrap utiliza CSS e JavaScript separados para estilização e funcionalidades. Isso significa que o HTML não precisa conter estilos ou scripts específicos, tornando-o mais limpo e fácil de manter.

Frameworks de UI:

O Bootstrap lida com a aparência e a interatividade via suas próprias regras CSS e plugins JS, enquanto o HTML continua focado na estrutura e no conteúdo.

4. Temas e Estilos Modulares

Temas Personalizáveis:

Bootstrap permite a personalização por meio de variáveis SCSS, o que possibilita alterar temas e estilos sem precisar alterar o HTML. Isso promove uma abordagem modular para personalização, mantendo a estrutura HTML intacta.

Override de Estilos:

Você pode adicionar estilos personalizados em CSS separado para ajustar a aparência sem modificar o HTML base.

Resumo

Bootstrap garante a independência estrutural do HTML ao fornecer um sistema de grid flexível, componentes reutilizáveis e estilos modulares, permitindo que você ajuste a aparência e o layout sem modificar a estrutura HTML. A separação entre CSS, JavaScript e HTML promove uma abordagem limpa e eficiente para o desenvolvimento web.

- ***Qual a relação entre o Bootstrap e a responsividade da página?***

Sistema de Grid Responsivo

Grid Flexível:

O Bootstrap usa um sistema de grid baseado em 12 colunas que se adapta automaticamente a diferentes tamanhos de tela. Isso permite criar layouts que se ajustam bem em dispositivos móveis, tablets e desktops.

Classes de Colunas:

Utiliza classes como `.col-md-6` e `.col-lg-4` para definir como as colunas devem se comportar em diferentes tamanhos de tela.

Media Queries Integradas

Breakpoints:

Bootstrap define breakpoints específicos para diferentes tamanhos de tela (como xs, sm, md, lg, xl). Isso permite aplicar estilos diferentes com base na largura da tela, garantindo que o design se ajuste de forma responsiva.

Classes Utilitárias:

Oferece classes utilitárias que ajudam a esconder, mostrar ou ajustar elementos com base no tamanho da tela.

Componentes Responsivos

Design Adaptável:

Muitos dos componentes do Bootstrap, como navegações, imagens e carrosséis, são projetados para se ajustar automaticamente às mudanças de tamanho da tela.

Classes de Layout:

Classes como `.container`, `.container-fluid`, e `.row` ajudam a criar layouts que se adaptam e se reconfiguram com base na tela do dispositivo.

Flexbox e Grid

Flexbox:

O Bootstrap utiliza o Flexbox para criar layouts flexíveis e responsivos. Isso permite que os elementos dentro de um contêiner se ajustem automaticamente, alinhando e distribuindo espaço de maneira eficiente.

CSS Grid:

As versões mais recentes do Bootstrap também oferecem suporte para CSS Grid, que proporciona mais controle sobre layouts complexos e responsivos.

Ferramentas de Personalização

Variáveis SCSS:

Permite personalizar breakpoints e outras configurações para ajustar o comportamento responsivo conforme necessário, mantendo a flexibilidade no design.

Resumo

Bootstrap facilita a criação de páginas responsivas por meio de seu sistema de grid flexível, media queries integradas, componentes adaptáveis e suporte para Flexbox e CSS Grid. Isso garante que o layout e os componentes da página se ajustem de forma adequada a diferentes tamanhos de tela, proporcionando uma experiência consistente e otimizada em diversos dispositivos.