

Procedimento 1 - Camadas de Persistência e Controle

Objetivos

O objetivo do Procedimento 1 foi o desenvolvimento de uma aplicação web baseada em **Jakarta EE**, denominada **CadastroEE**, possibilitando trabalhar com as camadas de persistência e controle; para tanto foi configurado todo um projeto na **IDE NetBeans**, utilizando as tecnologias **Jakarta EE**, com foco na construção de **entidades JPA**, criação de **Session Beans (EJB)** para operações *CRUD*, além da configuração de **conexão com banco de dados SQL Server** via *persistence.xml*.

Foi utilizado um servidor *GlassFish*, apoiado por um *Servlet*, encarregado por responder a requisições feitas por clientes, geralmente encaminhadas via **protocolo HTTP**.

A prática também incluiu a migração de código para o **namespace jakarta**, conforme as atualizações das especificações Jakarta EE 10.

Desta forma, a aplicação serve como base para sistemas empresariais que exigem persistência de dados, organização em camadas e acesso via interface 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

 Servlets

 HTML

 Ant (sistema de build)

 JDK 17

Estrutura do Projeto

O projeto obedeceu a uma estrutura organizada de forma a separar as responsabilidades em diferentes **módulos** ou **camadas**.

1. **Projeto Web (WAR):** Contém os **Servlets**, **JSPs**, **recursos estáticos** (imagens, CSS, JS), e o **arquivo de configuração web.xml**.

Função:

- a Controlar requisições e respostas HTTP através de Servlets.
 - b **Exibir páginas ao usuário**, sendo a camada de apresentação da aplicação.
 - c **Delegar as ações** para o EJB, que executa a lógica de negócio.
2. **Projeto EJB (JAR):** Contém **Session Beans**, as **entidades JPA** para persistência e as classes que compõem a **lógica de negócio**.

Função:

- a Implementar as regras de negócio, isolando a lógica da camada web.
 - b Efetuar a persistência em banco de dados usando Jakarta Persistence (**JPA**).
 - c Expor funcionalidades via **Session Beans**, possibilitando escalabilidade e reutilização de código.
3. **Configurações e Bibliotecas:**
 - a *persistence.xml*: Define a **unidade de persistência**, incluindo o datasource (ex: jdbc/loja) e as propriedades do provedor **JPA**.
 - b *web.xml*: Arquivo de configuração do **módulo web**, onde são definidos os mapeamentos de **Servlets**, filtros, **listeners**, tempo de sessão, etc.
 - c **Bibliotecas:**
 - **Jakarta EE 10 API** → adicionada ao projeto para suportar **Servlets**, **EJB**, **JPA**.
 - **GlassFish Server** → Servidor de aplicação para execução e testes.

Funcionalidades

A aplicação oferece as seguintes funcionalidades:

- ✓ Conexão Segura com o Banco de Dados.
- 📁 Modelagem e Mapeamento de Entidades
- 💰 Ajuste no Tipo de Dados
- ⚙️ Geração Automática de Session Beans
- 📦 Inclusão da Biblioteca Jakarta EE 10
- 🔄 Atualização das Importações
- 📝 Configuração da Persistência
- 🔄 Integração entre os módulos

[Client Browser]



[Servlet]



[Session Bean]



[JPA / DB]

Atividades Realizadas

- ✓ Configuração da conexão com SQL Server via JDBC.
- ✓ Ajuste do parâmetro *trustServerCertificate=true* para permitir conexões seguras.
- ✓ Criação das entidades de modelo, com destaque para Produto.
- ✓ Alteração do tipo do atributo precoVenda de *BigDecimal* para *Float*.
- ✓ Geração automática de *Session Beans* (EJB) com interfaces locais através do assistente "New → Session Beans for Entity Classes" no NetBeans.
- ✓ Inclusão da biblioteca **Jakarta EE 10** ao projeto.
- ✓ Atualização de todas as importações de *javax* para *jakarta*.
- ✓ Ajuste do arquivo *persistence.xml* conforme especificado.
- ✓ Modelagem da entidade Produto com anotações JPA;

- ✓ Criação da camada de persistência (ProdutoFacade) com uso do padrão Facade;
- ✓ Configuração do persistence.xml com datasource JTA para conexão com banco de dados;
- ✓ Desenvolvimento do ServletProduto para apresentar uma listagem dinâmica de produtos em HTML;
- ✓ Empacotamento dos módulos como .jar (EJB), .war (web) e .ear (distribuição final);
- ✓ Deploy e testes no servidor GlassFish.

Desafios Enfrentados

- ! Erro de conexão com o SQL Server
- ! Alteração de imports javax para jakarta
- ! Geração dos Session Beans (EJB) com o assistente.
- ! Erro no XML.
- ! Bloqueio de arquivos JAR.
- ! Configuração de datasource JDBC no servidor e ajuste de permissões.
- ! Necessidade de ajustes de compatibilidade entre versões do JDK e configurações do Ant.

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-1/CadastroEE.

Testes

Os testes foram realizados de forma manual, validando os seguintes pontos:

- Execução do servlet ServletProduto via navegador;
- Correta renderização da tabela HTML com dados do banco;
- Conectividade com o banco de dados via JPA;

- Testes com entidades reais no banco;
- Verificação dos logs do GlassFish em tempo real para identificação de erros.

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-1/RESULTADOS.pdf.

Conclusão

Esta prática proporcionou experiência na **configuração de ambientes corporativos** utilizando **Jakarta EE**, com atenção especial às mudanças na especificação.

O projeto "CadastroEE" foi concebido com uma arquitetura modular - foi implantado no servidor como um arquivo **.ear**, que inclui **.war** (web) e **.jar** (EJB) – o que facilita nas **manutenções futuras** e na **escalabilidade**.

O uso de tecnologias modernas e práticas como **EJB**, **JPA** e **Servlets** permitiu a separação clara entre camadas e responsabilidade de cada módulo, e facilitaram os testes.

Apesar dos desafios técnicos, a aplicação demonstrou robustez e eficiência.

As dificuldades enfrentadas foram superadas com pesquisa, entendimento da documentação oficial e apoio das ferramentas do próprio NetBeans.

A prática proporcionou uma visão completa sobre o ciclo de desenvolvimento, desde a modelagem até a implantação em um servidor de aplicações.

O projeto finalizou com sucesso, possibilitando a realização de operações **CRUD**, com persistência em banco de dados.

O exercício permitiu consolidar conhecimentos técnicos importantes, como:

- ✓ **Programação Java** aplicada a sistemas corporativos.
- ✓ **Criação e configuração de Servlets** para manipular requisições HTTP.
- ✓ **Desenvolvimento de componentes EJB** para encapsular regras de negócio.

- ✓ **Mapeamento de entidades JPA** e persistência de dados em **SQL Server**.
- ✓ **Configuração de JDBC Driver** para estabelecer conexões com o banco de dados.
- ✓ **Integração de módulos** em um ambiente **Jakarta EE 10**, utilizando **NetBeans**.
- ✓ **Configuração de servidores de aplicação** como **GlassFish**.

? Questionamentos

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

Um **projeto corporativo** no NetBeans, especialmente quando voltado para **Java EE/Jakarta EE**, é geralmente estruturado de forma **modular**, permitindo separação de responsabilidades. Os principais módulos são:

- **Módulo EJB (Enterprise JavaBeans)**: Contém a lógica de negócios. Inclui Session Beans, entidades JPA, etc.
- **Módulo Web**: Responsável pela interface com o usuário (Servlets, JSP, JSF).
- **Módulo EAR (Enterprise Archive)**: Empacota os módulos EJB e Web em um único arquivo .ear para implantação em servidores como GlassFish ou WildFly.
- **Bibliotecas compartilhadas**: Utilizadas por múltiplos módulos, como arquivos JAR de dependências externas.

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

- **JPA (Java Persistence API)**:
 - Responsável pela **persistência de dados** (interação com banco de dados).
 - Mapeia objetos Java para tabelas do banco de dados (ORM).
 - Facilita operações CRUD sem a necessidade de SQL explícito.
- **EJB (Enterprise JavaBeans)**:
 - Fornece uma camada de negócios transacional, segura e escalável.
 - Os **Session Beans** encapsulam lógica de negócio e são gerenciados pelo container EJB.

- Permite transações distribuídas, segurança declarativa, e injeção de dependência.

Juntos, EJB e JPA formam a base do *backend* de um sistema corporativo Java EE.

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

O NetBeans melhora a produtividade através de várias facilidades incorporadas, tais como:

- Assistentes para criação de entidades **JPA** a partir de tabelas do banco de dados.
- Geração automática de facades **EJB** para entidades JPA (métodos CRUD prontos).
- Refatoração assistida, completamento de código, e validação em tempo real.
- Gerenciamento visual de persistência e EJBs.
- Deploy automático em servidores integrados.
- Suporte a **anotações JPA e EJB**, com sugestões e validações automáticas.

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 rodam no servidor e tratam requisições HTTP. Fazem parte da especificação Java EE e são utilizados para controlar o fluxo de dados entre a camada de apresentação (navegador) e a lógica de negócios.

O **NetBeans** oferece suporte a Servlets por meio de:

- Assistente de criação de **Servlet** com geração de código básico.
- Suporte ao mapeamento de URLs via anotação (**@WebServlet**) ou via *web.xml*.
- Execução e debug automático em servidores integrados.
- Auto-complete e validação para métodos como *doGet*, *doPost*, etc.

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

A comunicação entre **Servlets (Web layer)** e **Session Beans (EJBs - Business layer)** é feita por meio de *injeção de dependência*, utilizando a anotação `@EJB` dentro do Servlet.

Essa injeção permite que o container EJB entregue automaticamente uma instância gerenciada do Session Bean ao Servlet. Isso garante:

- Gerenciamento de transações.
- Controle de concorrência.
- *Pooling* eficiente de recursos.