**\*Módulo 1: Tecnologias JPA e JEE\***

Este módulo introduz as tecnologias centrais do JEE (Java Enterprise Edition) para desenvolvimento web: JPA (Java Persistence API) e EJB (Enterprise JavaBeans) [1]. Você aprenderá a utilizar essas tecnologias para descrever e organizar a arquitetura MVC (Model, View e Controller), além de construir um exemplo completo utilizando o padrão Front Controller.

**\* \*Preparação do ambiente:\*** Instalação do JDK, Apache NetBeans, Web Server Tomcat e Application Server GlassFish. A sugestão é configurar a porta do Tomcat como 8084 para evitar conflitos com o GlassFish na porta 8080.

**\* \*Objetivos:\*** Descrever as características do JPA, empregar componentes EJB na construção de regras de negócio, descrever a utilização da arquitetura MVC na plataforma Java e empregar o padrão Front Controller em sistema MVC com interface Java Web.

**\* \*JPA (Java Persistence API):\*** Abordado para o mapeamento objeto-relacional [3]. Diferencia a estrutura de bancos de dados relacionais (tabelas, registros, chaves primárias/estrangeiras) da programação orientada a objetos (classes, instâncias, coleções, atributos). O mapeamento objeto-relacional (ORM) permite que aplicações orientadas a objetos se comuniquem com bancos de dados. A técnica ORM preenche uma entidade (objeto) com dados de um registro (relacional). Frameworks de persistência, geralmente baseados em XML ou anotações, geram automaticamente os comandos SQL. O JPA padroniza a arquitetura dos frameworks de persistência e concentra as configurações no arquivo persistence.xml, configurável por meio de anotações. A JPA tem o padrão DAO implícito, o que aumenta a eficiência na persistência.

**\* \*Entidade JPA:\*** Uma classe sem métodos de negócios (POJO) que recebe anotações para mapeamento entre a classe e sua tabela. Anotações importantes incluem @Entity, @Table, @NamedQueries, @Id, @Basic, @Column, @OneToMany, @ManyToOne, @OneToOne, @ManyToMany, @OrderBy, @JoinColumn. Entidades JPA devem ter um construtor vazio e outro baseado na chave primária, além dos métodos equals e hashCode. O atributo serialVersionUID é utilizado em processos de migração de base de dados.

**\* \*Arquivo persistence.xml:\*** Configura aspectos gerais da conexão com o banco de dados, definindo a unidade de persistência, o tipo de transação e o provedor de persistência.

**\* \*EntityManager:\*** Concentra os métodos que invocam os comandos SQL montados pelo JPA de forma transparente. É obtido a partir de um EntityManagerFactory.

**\* \*Consulta e Manipulação de Dados:\*** Exemplos de código demonstram a consulta de dados usando createNamedQuery e getResultList, inclusão de dados usando persist dentro de uma transação, alteração de dados usando merge em uma transação, e exclusão de dados usando find e remove em uma transação. Os métodos find, persist, merge e remove correspondem a SELECT, INSERT, UPDATE e DELETE no banco de dados. O JPA não elimina o uso de JDBC, apenas automatiza a geração de comandos SQL.

**\* \*Configuração no NetBeans:\*** Adicionar a biblioteca JDBC do Derby (Java DB) e o framework EclipseLink. Criar um banco de dados Derby na aba Services do NetBeans e executar comandos SQL para criar tabelas e inserir dados.

**\* \*NamedQueries com Parâmetros:\*** Demonstra como utilizar parâmetros em NamedQueries para recuperar registros específicos.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**\*Módulo 2: Implementação de regras de negócio com EJBs\***

Este módulo foca na implementação de regras de negócio utilizando Enterprise Java Beans (EJB), componentes corporativos no ambiente Java.

**\* \*Enterprise Java Beans (EJB):\*** Componentes corporativos utilizados em servidores de aplicação como GlassFish, suportando elementos da plataforma JEE. Executados em um pool de objetos. Podem ser acessados via interface local (EJBLocalObject) ou remota (EJBObject). A programação no JEE5+ é simplificada pelo uso de anotações.

**\* \*Tipos de Session Beans:\*** Componentes fundamentais para encapsular a lógica de negócios e fornecer serviços, utilizados para implementar transações, acesso a bancos de dados e outras operações relacionadas à lógica de negócios. Existem três tipos:

**\* \*Stateless:\*** Não mantém estado entre chamadas sucessivas, sendo ágil.

**\* \*Stateful:\*** Utilizado quando é necessário manter valores entre chamadas sucessivas (ex: cesta de compras).

**\* \*Singleton:\*** Permite apenas uma instância por máquina virtual, garantindo o compartilhamento de dados entre usuários.

**\* \*Interface de Acesso:\*** Definida antes do session bean com anotações @Local (acesso interno) ou @Remote (acesso remoto). O acesso local é suficiente para acionar EJBs a partir de servlets.

**\* \*Session Bean com Servlet:\*** Demonstra como utilizar um session bean a partir de um Servlet anotando um atributo com @EJB.

**\* \*Message-driven beans (MDB):\*** Tipo de EJB para comunicação com mensagerias via JMS (Java Message Service), possibilitando processamento assíncrono. Permite trabalhar nos modelos publish/subscribe e point to point. As mensagerias permitem construir sistemas B2B com baixo acoplamento. A criação de filas ou tópicos no GlassFish pode ser feita com o comando asadmin. O MDB é definido com a anotação MessageDriven e implementa a interface MessageListener, com o método onMessage tratando a recepção de mensagens. MDBs não podem ser acionados diretamente como session beans; sua ativação ocorre quando um cliente posta uma mensagem. O processo de postagem de mensagem para um MDB a partir de um Servlet é demonstrado utilizando anotações @Resource para mapear a fábrica de conexões e a fila de destino.

**\* \*Aplicativo Corporativo:\*** Softwares desenvolvidos para atender necessidades complexas utilizando tecnologias Java EE, fornecendo recursos como gerenciamento de transações, segurança, escalabilidade e integração com sistemas legados. A estrutura de um aplicativo corporativo no NetBeans pode ser Enterprise Application, gerando projetos com extensões "jar" (EJBs e entidades JPA), "war" (elementos web) e "ear" (agrupando os anteriores para implantação). A implantação deve ser feita no projeto principal com extensão ear. A criação de session beans e MDBs dentro do projeto corporativo no NetBeans é detalhada .

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Texto, Carta

O conteúdo gerado por IA pode estar incorreto.

**\*Módulo 3: Arquitetura MVC no Java\***

Este módulo aborda a arquitetura MVC (Model-View-Controller) como um padrão arquitetural importante em sistemas corporativos.

**\* \*Padrões de Desenvolvimento:\*** Fornecem soluções reutilizáveis para problemas comuns na programação. Padrões destacados em sistemas corporativos incluem Facade, Proxy, Flyweight, Front Controller e DAO. O padrão DAO é observado na tecnologia JPA. Descrições formais de diversos padrões são apresentadas.

**\* \*Padrões Arquiteturais:\*** Definem a estrutura de alto nível de um sistema, formalizando a organização em termos de componentes e interações. Existem diferentes definições de modelos para os padrões arquiteturais. O MVC é um padrão arquitetural para sistemas interativos. Sistemas corporativos complexos combinam múltiplos padrões, com o MVC sendo comum em sistemas cadastrais.

**\* \*Arquitetura MVC (Model-View-Controller):\*** Divide o sistema em três camadas com responsabilidades específicas.

**\* \*Model:\*** Contém entidades e classes para acesso ao banco de dados, controlando a persistência e encapsulando o estado do sistema. Pode utilizar mapeamento objeto-relacional e o padrão DAO é aplicável.

**\* \*Controller:\*** Implementa as regras de negócio, solicita dados à camada Model. Não pode ser direcionada para uma interface. Pode utilizar objetos distribuídos e o padrão Facade facilita sua utilização.

**\* \*View:\*** Define a interface do sistema, faz requisições para a camada Controller e contém apenas regras de formatação. Podem ser definidas múltiplas interfaces e não pode acessar a camada Model.

**\* \*Regras Fundamentais do MVC:\*** Elementos da View não podem acessar a camada Model; apenas objetos de negócio do Controller podem acessar o Model. A arquitetura é baseada em camadas, onde cada camada "enxerga" apenas a camada imediatamente abaixo. Entidades são unidades de informação para trânsito entre as camadas. A camada Controller deve ser independente de ambiente específico. O Controller é o local ideal para definir regras de autorização.

**\* \*Componentes Java para MVC:\*** Diversos frameworks auxiliam na implementação do MVC, como JSF (View), Spring ou EJB (Controller) e Hibernate (Model/JPA). O uso de camadas especializadas permite a divisão da equipe. No contexto do material, a camada Model utiliza JPA e a camada Controller utiliza EJB, definidas no mesmo projeto. Essa abordagem garante a independência do núcleo funcional e lógico do sistema. Definir controladores como servlets confunde regras de negócio com rotinas de conversão HTTP, o que é uma abordagem errônea.

**\* \*Implementação de Exemplo:\*** Demonstra a criação da entidade Produto com JPA no projeto EJB e um session bean ProdutoGestor (Controller) para operações cadastrais, utilizando um EntityManagerFactory e métodos JPA. O arquivo persistence.xml é adicionado ao projeto EJB. Um Servlet ServletListaProduto (View) é criado no projeto WAR para exibir a listagem de produtos, acessando o session bean ProdutoGestor via @EJB. Um exemplo mais complexo com relacionamento entre entidades (Empresa e Produto) é apresentado, incluindo a criação das entidades, um EJB ProdutoGestor modificado para lidar com o relacionamento, e um Servlet para listar produtos exibindo informações da empresa associada.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Texto, Carta

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Texto, Carta

O conteúdo gerado por IA pode estar incorreto.

**\*Módulo 4: Padrão Front Controller em sistemas MVC\***

Este módulo detalha a implementação do padrão Front Controller em sistemas MVC, com foco na camada View.

**\* \*Padrão Front Controller:\*** Centraliza as chamadas em um único ponto de acesso, direcionando os resultados para a interface correta. Sua implementação deve ocorrer na camada View. Em aplicativos Java para Web, pode ser implementado com base em um servlet. O processo envolve a recepção de uma chamada HTTP, execução de operações que envolvam chamadas aos EJBs e redirecionamento para uma página (JSP).

**\* \*Exemplo Prático:\*** Criação de um projeto Enterprise Application (CadastroEJB) utilizando GlassFish e Java EE 7. As tabelas EMPRESA, DEPARTAMENTO e SERIAIS são definidas no banco de dados Derby para um cadastro simples com relacionamento e autoincremento. As cinco páginas da interface (index.html, ListaEmpresa.jsp, DadosEmpresa.jsp, ListaDepartamento.jsp, DadosDepartamento.jsp) são descritas. O fluxo das chamadas HTTP é modelado com uma Rede de Petri. Um parâmetro (acao) é utilizado para identificar a transição requerida, com valores correspondentes às operações.

**\* \*Camadas Model e Controller para o Exemplo:\*** Criadas no projeto EJB (CadastroEJB-ejb) usando JPA e EJB.

**\* \*Criação da Camada Model:\*** Utiliza ferramentas do NetBeans para gerar entidades a partir do banco de dados (Departamento e Empresa). Modificações são feitas nas entidades para configurar o uso de autoincremento baseado em tabela (TableGenerator).

**\* \*Criação da Camada Controller:\*** Utiliza ferramentas do NetBeans para gerar Session Beans (Facade) para as entidades. Uma classe abstrata AbstractFacade é gerada para concentrar processos genéricos de acesso e manipulação de dados. Session beans específicos (DepartamentoFacade e EmpresaFacade) herdam de AbstractFacade e utilizam anotações @Stateless e @PersistenceContext. Interfaces de acesso local (DepartamentoFacadeLocal e EmpresaFacadeLocal) são definidas. O controle transacional utiliza JTA. Ajustes no persistence.xml e glassfish-resources.xml podem ser necessários devido a convenções de nomes.

**\* \*Camada View para o Exemplo:\*** Construída no projeto WAR (CadastroEJB-war).

**\* \*Criação das Páginas JSP:\*** Criação de DadosEmpresa.jsp (formulário simples), DadosDepartamento.jsp (formulário com seleção de empresa), ListaEmpresa.jsp (listagem de empresas com links para inclusão e exclusão), e ListaDepartamento.jsp (listagem de departamentos com links para inclusão e exclusão, exibindo informações da empresa). O arquivo index.html é modificado para conter links para as listagens.

**\* \*Implementação do Front Controller:\*** Utiliza um servlet (CadastroFC) no projeto WAR e um padrão Strategy para segmentar as chamadas aos EJBs. Classes Strategy (abstrata), EmpresaStrategy, e DepartamentoStrategy são criadas. O servlet CadastroFC centraliza as requisições, utiliza @EJB para acessar os facades, e direciona a execução para a estratégia apropriada com base no parâmetro acao. HashMaps são utilizados para armazenar as estratégias e relacionar ações com identificadores de gestores.

**\* \*Aplicação MVC com Padrão Front Controller**:\* Exercício prático para simular uma aplicação de livraria online, implementando o padrão Front Controller e MVC com EJBs para a camada Model. Classes como FrontController, Controller (interface), HomeController, BookController e Book são apresentadas. O FrontController mapeia ações para controladores. O BookController acessa um EJB (Book) para obter a lista de livros.

Este resumo abrange os principais conceitos e exemplos práticos apresentados no material, que são cruciais para a compreensão e aplicação das tecnologias JPA, EJB e da arquitetura MVC com padrão Front Controller em sistemas Java EE.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto. Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Exercícios:

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.