Nome **: Angemydelson Saint-Bert**
Matrícula **: 2121101002**
Prof **: Felipe Grando**
Disciplina **: Programação II**

Explicação do código

- **CadastroProductController.java**

A classe "CadastroProductController" é uma classe que gerencia a página de cadastro de produtos em uma aplicação Java Web. Essa classe é uma ManagedBean, gerenciada pelo JSF/EJB, que lida com as requisições do usuário e interage com a camada de dados.
A classe possui o escopo "ViewScoped", o que significa que o objeto permanecerá ativo enquanto não houver um redirecionamento para outra página. O escopo "ViewScoped" é útil quando é necessário que o objeto permaneça ativo durante uma operação do usuário em uma determinada página.
A classe possui vários atributos, como "FacesContext", "ProductDAO" e "TipoPermissaoDAO", que são injetados pelo CDI (Contexts and Dependency Injection) e são usados para gerenciar o acesso aos dados da aplicação e a exibição de mensagens na tela do usuário.
A classe possui um método chamado "init()" que é anotado com "@PostConstruct". Esse método é executado após a classe ser instanciada e todos os seus atributos serem injetados. Nesse método, é realizada a verificação da permissão do usuário e a inicialização dos atributos importantes da classe.
A classe possui um método chamado "novoCadastro()" que é chamado ao clicar no botão "Novo" na tela de cadastro de produtos. Esse método simplesmente cria um novo objeto do tipo "Product".
A classe possui um método chamado "salvar()" que é chamado ao clicar no botão "Salvar" na tela de cadastro de produtos. Esse método é responsável por verificar se os dados do produto são válidos (de acordo com as regras de negócios) e, em seguida, salvar ou atualizar o produto no banco de dados. Para isso, é utilizado o objeto "ProductDAO" injetado pelo CDI.
Além disso, a classe possui outros atributos e métodos que são responsáveis por lidar com a interação do usuário na tela, como "Product" (objeto que representa o produto que está sendo cadastrado), "listaProducts" (lista de todos os produtos cadastrados), "permissoes" (lista de permissões que podem ser atribuídas ao produto) e "permissoesSelecionadas" (lista das permissões selecionadas pelo usuário). Esses atributos e métodos são utilizados para preencher e exibir informações na tela do usuário.

- **CadastroUsuarioController.java**

A classe CadastroUsuarioController é responsável por gerenciar as ações relacionadas ao cadastro de usuários em uma aplicação web. Ela é uma classe Java que segue o padrão

Model-View-Controller (MVC), onde é a responsabilidade do controller intermediar as requisições do usuário com o modelo de dados (model) e apresentá-las na camada de visualização (view).
Aqui está uma explicação detalhada dos atributos e métodos da classe:
Atributos:

- facesContext: objeto responsável por fornecer informações sobre o contexto da aplicação. É injetado pelo CDI (Contexts and Dependency Injection).
- passwordHash: objeto responsável por aplicar o algoritmo de hash (Pbkdf2) na senha do usuário. É injetado pelo CDI.
- usuarioDAO: objeto responsável por gerenciar a persistência de dados do usuário. É injetado pelo CDI.
- tipoPermissaoDAO: objeto responsável por gerenciar a persistência de dados das permissões de usuários. É injetado pelo CDI.
- usuario: objeto que representa o usuário a ser cadastrado ou atualizado.
- listaUsuarios: lista de usuários cadastrados na aplicação.
- permissoes: lista de permissões de usuários disponíveis na aplicação.
- permissoesSelecionadas: lista de permissões selecionadas para o usuário em tela.
- termoDePesquisa: termo utilizado para filtrar a lista de usuários cadastrados na aplicação.

Métodos:

- init(): método que é executado após a instância da classe ser criada. Ele verifica se o usuário está autenticado e possui permissão adequada para acessar o cadastro de usuários. Em seguida, inicializa as listas de permissões, usuários e permissões selecionadas para o usuário em tela.
- novoCadastro(): método chamado pelo botão "Novo" para limpar os campos de cadastro de usuário e habilitar a edição.
- salvar(): método chamado pelo botão "Salvar" para persistir os dados do usuário. Ele primeiro verifica se o usuário é válido (regras de negócio). Em seguida, limpa a lista de permissões de usuário e adiciona as permissões selecionadas em tela. Depois, aplica o algoritmo de hash na senha do usuário e verifica se é um novo cadastro ou atualização de cadastro. Por fim, recarrega a lista de usuários para atualizar a tabela com os novos dados.
- excluir(): método chamado pelo botão "Excluir" para remover um usuário da lista de usuários. Ele verifica se o usuário é válido (regras de negócio) e, em seguida, o remove da lista de usuários e persiste a alteração.
- usuarioValido(): método que verifica se o usuário é válido. Ele verifica se o nome, sobrenome, email e senha foram preenchidos corretamente. Se alguma informação estiver faltando ou incorreta, uma mensagem de erro é exibida na tela.
- buscar(): método chamado pelo botão "Buscar" para filtrar a lista de usuários por um termo de pesquisa. Ele atualiza a lista de usuários com aqueles que contém o termo de pesquisa em seu nome ou sobrenome.

ProductDAO

- ➢ **ProductDAO.java**

A classe ProductDAO é uma classe Java que implementa as regras de negócio para operações de persistência de dados em um banco de dados relacional. Ela é responsável por criar, atualizar, excluir e buscar objetos da classe Product no banco de dados.
A seguir, explicaremos cada método da classe ProductDAO em detalhes:

1. encontrarId(Integer id): Este método recebe um parâmetro inteiro id e retorna o objeto Product correspondente ao ID fornecido. Ele usa o método find() do objeto EntityManager para pesquisar o banco de dados e retornar o objeto Product correspondente.
2. ehProductUnico(String u): Este método verifica se o valor da propriedade product do objeto Product é único no banco de dados. Ele usa a API Criteria do Hibernate para criar uma consulta que seleciona todos os objetos Product cuja propriedade product é semelhante ao valor fornecido no parâmetro u. Se a lista resultante estiver vazia, o método retorna true, caso contrário, retorna false.
3. listarTodos(): Este método retorna uma lista contendo todos os objetos Product no banco de dados. Ele usa a linguagem HQL (Hibernate Query Language) para criar uma consulta que seleciona todos os objetos Product.
4. salvar(Product u): Este método recebe um objeto Product como parâmetro e o salva no banco de dados. Ele usa o método persist() do objeto EntityManager para salvar o objeto.
5. atualizar(Product u): Este método recebe um objeto Product como parâmetro e o atualiza no banco de dados. Ele usa o método merge() do objeto EntityManager para atualizar o objeto.
6. excluir(Product u): Este método recebe um objeto Product como parâmetro e o remove do banco de dados. Ele usa o método remove() do objeto EntityManager para excluir o objeto.
7. buscarPorTermo(String termo): Este método recebe um parâmetro termo e retorna uma lista de objetos Product cuja propriedade nome contém o valor do parâmetro. Ele usa a API Criteria do Hibernate para criar uma consulta que seleciona todos os objetos Product cuja propriedade nome é semelhante ao valor fornecido no parâmetro termo.

➢ **TipoPermissaoDAO.java**

A classe TipoPermissaoDAO é responsável por realizar operações de persistência de dados relacionadas à entidade TipoPermissao. A seguir, cada um dos métodos presentes na classe é explicado detalhadamente:

- encontrarPermissao(Integer permissaoId): esse método recebe como parâmetro um id de permissão e retorna a instância de TipoPermissao correspondente a esse id. Ele utiliza o método find do EntityManager para buscar a entidade no banco de dados.
- listarTodos(): esse método retorna uma lista com todas as instâncias de TipoPermissao cadastradas no banco de dados. Ele utiliza uma query em JPQL para buscar todas as entidades.

Ambos os métodos são simples e não apresentam complexidade em sua implementação. Além disso, a classe é anotada com @Stateful, o que indica que o bean criado a partir dessa classe terá um estado associado e poderá ser mantido por toda a vida da aplicação. O

EntityManager é injetado na classe através da anotação @Inject, que permite a utilização de injeção de dependência para obter a instância do EntityManager a partir do container do EJB.

➢ **Product.java**

A classe "Product" é uma classe modelo que representa um produto e contém informações como nome, código, quantidade, preço e permissões relacionadas a esse produto. A seguir, uma explicação mais detalhada dos métodos da classe:

- getId(): método getter para o atributo id, que representa a chave primária da tabela de produtos.
- setId(Integer id): método setter para o atributo id, que define o valor da chave primária da tabela de produtos.
- getNome(): método getter para o atributo nome, que representa o nome do produto.
- setNome(String nome): método setter para o atributo nome, que define o valor do nome do produto.
- getCodigo(): método getter para o atributo codigo, que representa o código do produto.
- setCodigo(Integer codigo): método setter para o atributo codigo, que define o valor do código do produto.
- getQuantitykl(): método getter para o atributo quantitykl, que representa a quantidade do produto em quilogramas.
- setQuantitykl(Double quantitykl): método setter para o atributo quantitykl, que define o valor da quantidade do produto em quilogramas.
- getQuantityint(): método getter para o atributo quantityint, que representa a quantidade do produto em unidades.
- setQuantityint(Double quantityint): método setter para o atributo quantityint, que define o valor da quantidade do produto em unidades.
- getPriceund(): método getter para o atributo priceund, que representa o preço unitário do produto.
- setPriceund(Double priceund): método setter para o atributo priceund, que define o valor do preço unitário do produto.
- getPricekl(): método getter para o atributo pricekl, que representa o preço por quilograma do produto.
- setPricekl(Double pricekl): método setter para o atributo pricekl, que define o valor do preço por quilograma do produto.
- getQuantidadeTotalemEstoque(): método getter para o atributo quantidadeTotalemEstoque, que representa a quantidade total do produto em estoque.
- setQuantidadeTotalemEstoque(Double quantidadeTotalemEstoque): método setter para o atributo quantidadeTotalemEstoque, que define o valor da quantidade total do produto em estoque.
- getPrecoTotalemEstoque(): método getter para o atributo precoTotalemEstoque, que representa o preço total do produto em estoque.
- setPrecoTotalemEstoque(Double precoTotalemEstoque): método setter para o atributo precoTotalemEstoque, que define o valor do preço total do produto em estoque.

- setPermissoes(List<TipoPermissao> permissoes): método setter para o atributo permissoes, que define a lista de permissões relacionadas ao produto.
- getPermissoes(): método getter para o atributo permissoes, que retorna a lista de permissões relacionadas ao produto.

➢ **TipoPermissao.java**

A classe TipoPermissao é uma entidade que representa os diferentes tipos de permissões que podem ser atribuídos a um usuário em um sistema. A seguir, serão detalhados cada um dos métodos presentes na classe:

- getId() e setId(Integer id): Esses métodos são getters e setters, respectivamente, do atributo id. O atributo id é uma chave primária da entidade TipoPermissao e é gerado automaticamente pelo banco de dados a cada nova inserção na tabela tipo_permissao.
- getPermissao() e setPermissao(Permissao permissao): Esses métodos são getters e setters, respectivamente, do atributo permissao. O atributo permissao é do tipo Permissao, que é uma enumeração que define os diferentes tipos de permissões que podem ser atribuídas a um usuário.
- getUsuarios(): Esse método retorna uma lista de usuários que possuem a permissão representada pela instância atual de TipoPermissao.
- addUsuario(Usuario usuario): Esse método adiciona um usuário à lista de usuários que possuem a permissão representada pela instância atual de TipoPermissao. Além disso, esse método também adiciona a instância atual de TipoPermissao à lista de permissões do usuário adicionado, estabelecendo assim o relacionamento ManyToMany entre Usuario e TipoPermissao.
- getProducts() e getVendas(): Esses métodos retornam, respectivamente, listas de produtos e vendas que possuem a permissão representada pela instância atual de TipoPermissao. No entanto, esses métodos estão comentados e, portanto, não são utilizados na classe.
- addProduct(Product product) e addVenda(Venda venda): Esses métodos adicionam, respectivamente, um produto e uma venda à lista de produtos e vendas que possuem a permissão representada pela instância atual de TipoPermissao. Assim como o método addUsuario(Usuario usuario), esses métodos também estabelecem o relacionamento ManyToMany entre Product/Venda e TipoPermissao. No entanto, esses métodos também estão comentados e, portanto, não são utilizados na classe.

➢ **AppConfig.java**

A classe AppConfig contém as configurações de autenticação e autorização para a API de segurança do projeto. Essas configurações são feitas através de anotações.

A anotação @CustomFormAuthenticationMechanismDefinition é usada para definir a página de login e a página de erro que serão exibidas quando o usuário tentar acessar uma página que requer autenticação. No caso dessa classe, as páginas definidas são login.xhtml e login-error.xhtml.

A anotação @DatabaseIdentityStoreDefinition é usada para definir o local e a forma como as informações de usuários e permissões serão consultadas e autenticadas. Nessa classe, é definido que as informações serão consultadas em um banco de dados Postgres, através do dataSourceLookup definido como java:/PostgresDS. O parâmetro callerQuery define a consulta SQL que será usada para buscar a senha do usuário. Já o parâmetro groupsQuery define a consulta SQL que será usada para buscar as permissões do usuário. Essas permissões serão obtidas a partir do relacionamento entre as tabelas tipo_permissao, permissao e usuario.
Por fim, a anotação @ApplicationScoped indica que essa classe será gerenciada pelo contexto da aplicação e que uma única instância será compartilhada por todos os usuários.

- ➢ **LoginController.java**

A classe LoginController é um controlador de página que gerencia as operações de login e logout do usuário. Ele é usado em conjunto com a classe AppConfig para autenticar e autorizar usuários na aplicação web.
Aqui estão as explicações detalhadas dos métodos dessa classe:

1. inicializarUsuario(): método chamado após a instância da classe LoginController. Ele cria uma nova instância de Usuario para armazenar as credenciais do usuário durante o processo de login.
2. login(): método que realiza a autenticação do usuário. Primeiro, ele verifica se já existe um usuário autenticado na sessão e exibe uma mensagem de erro caso exista. Caso contrário, ele cria uma instância de UsernamePasswordCredential com as credenciais fornecidas pelo usuário. Em seguida, ele chama o método securityContext.authenticate para autenticar o usuário com as credenciais fornecidas. Se a autenticação for bem-sucedida, o método redireciona o usuário para a página de cadastro de usuários ou de produtos, dependendo do tipo de usuário autenticado. Se a autenticação falhar, uma mensagem de erro é exibida.
3. logout(): método que efetua o logout do usuário. Ele invalida a sessão do usuário e redireciona o usuário para a página de logout.
4. getUsuario(): método que retorna o objeto Usuario atualmente armazenado na instância da classe.
5. setUsuario(): método que atualiza o objeto Usuario armazenado na instância da classe com um novo valor.

- ➢ **AdminSetup.java**

A classe "AdminSetup" é um ServletContextListener que é acionado uma única vez quando a aplicação é iniciada no servidor. O objetivo dessa classe é criar um usuário administrador padrão para a aplicação, caso ele ainda não exista.
A seguir, cada método da classe é explicado em detalhes:
Método "contextInitialized":

1. Esse método é acionado pelo container web quando a aplicação é iniciada. Ele é responsável por verificar se o usuário administrador já foi criado. Se o usuário ainda não existir, o método cria um novo usuário administrador com o email, cidade, país, CPF, senha, usuário e permissões padrão. As informações de login padrão são definidas como "admin" e a senha padrão é criptografada usando um hash

Pbkdf2PasswordHash. O método também salva o novo usuário no banco de dados usando o método "salvar" da classe "UsuarioDAO".

Injeção de dependências:

2. A classe faz uso de três classes: Pbkdf2PasswordHash, UsuarioDAO e TipoPermissaoDAO. Essas classes são injetadas usando a anotação @Inject, permitindo que a classe AdminSetup tenha acesso a esses objetos.

Variáveis de instância:

3. A classe possui três variáveis de instância: "passwordHash", "usuarioDAO" e "tipoPermissaoDAO". Elas são usadas para acessar as classes injetadas e executar as operações necessárias.

Variável "admin":

4. A variável "admin" é uma instância da classe "Usuario" que representa o usuário administrador padrão. O objeto é criado se o usuário ainda não existir.

Método "ehUsuarioUnico":

5. Esse método é definido na classe "UsuarioDAO" e é usado para verificar se o usuário administrador já foi criado. Ele retorna true se o usuário não existe no banco de dados.

Método "encontrarPermissao":

6. Esse método é definido na classe "TipoPermissaoDAO" e é usado para obter o tipo de permissão para o usuário administrador padrão. O tipo de permissão é definido como "ADMINISTRADOR".

Método "generate":

7. Esse método é definido na classe "Pbkdf2PasswordHash" e é usado para gerar um hash seguro da senha padrão do usuário administrador.

Método "addUsuario":

8. Esse método é definido na classe "TipoPermissao" e é usado para adicionar o usuário administrador padrão às permissões do tipo de permissão correspondente.

Método "salvar":

9. Esse método é definido na classe "UsuarioDAO" e é usado para salvar o usuário administrador padrão no banco de dados. O método recebe o objeto "admin" como parâmetro.

➢ **Permissao.java**

A classe Permissao é uma enumeração que define três constantes com valores inteiros associados: ADMINISTRADOR, CLIENTE e SERVIDOR.

Cada constante é criada usando o construtor da classe, que recebe um parâmetro id do tipo int. Esse id é usado para identificar o tipo de permissão na base de dados do sistema.

A enumeração Permissao é usada em outras classes do projeto para determinar o nível de acesso de usuários em diferentes partes do sistema, de acordo com sua permissão. Por exemplo, um usuário com permissão ADMINISTRADOR teria acesso a funcionalidades e recursos que um usuário com permissão CLIENTE não teria.

Essa classe é usada para criar um tipo customizado/complexo, para representar de forma mais clara as permissões dos usuários em um sistema. Como são constantes, elas não podem ser alteradas durante a execução do programa.

➢ **Resources.java**

A classe Resources é responsável por produzir e gerenciar objetos importantes para a aplicação, utilizando a anotação @Produces do CDI (Contexts and Dependency Injection). Essa classe possui dois métodos que são explicados a seguir:

- private EntityManager em: Esse método produz uma instância do objeto EntityManager, que é uma interface utilizada pelo JPA (Java Persistence API) para gerenciar a comunicação com o banco de dados. O JPA permite que os objetos Java sejam armazenados em um banco de dados relacional e que esses objetos sejam recuperados posteriormente. A anotação @PersistenceContext indica que o contexto de persistência será gerenciado pelo contêiner EJB, o que significa que o contêiner irá gerenciar a transação e a sincronização com o banco de dados.
- public FacesContext produceFacesContext(): Esse método produz uma instância do objeto FacesContext, que é uma classe que representa o contexto atual do JSF (JavaServer Faces). O FacesContext é um objeto muito importante no JSF, pois contém informações sobre o estado atual da aplicação, o ciclo de vida da requisição, e outros objetos importantes, como o ExternalContext e o ViewRoot. A anotação @RequestScoped indica que a instância do objeto FacesContext será criada e gerenciada pelo contêiner para cada requisição HTTP, ou seja, a instância será criada no início da requisição e destruída no final da mesma.

➢ **cadastro_product.xhtml**

Essa classe JavaServer Faces (JSF) com a tag cadastro_product é responsável pela exibição e interação com o formulário de cadastro de produtos. O objetivo é permitir ao usuário inserir, atualizar e excluir produtos no sistema.

Aqui está uma explicação detalhada das tags usadas nessa classe:

- xmlns: define o namespace do XHTML usado na página;
- xmlns:ui: define o namespace do Facelets usado na página;
- xmlns:f: define o namespace do JSF usado na página;
- xmlns:h: define o namespace das tags HTML padrão do JSF usadas na página;
- xmlns:p: define o namespace das tags PrimeFaces usadas na página;
- template: define o template usado pela página, que nesse caso é o default.xhtml;
- ui:composition: essa é a tag raiz da página e ela define que a página é uma composição de outros elementos;
- ui:define name="content": define um componente chamado "content" que será injetado no template como conteúdo principal;
- h:form: define um formulário HTML padrão do JSF;
- p:inputText: define um campo de entrada de texto do PrimeFaces;
- p:commandButton: define um botão de comando do PrimeFaces que pode realizar uma ação, como fazer uma busca ou salvar um produto;
- p:growl: define uma caixa de mensagens do PrimeFaces que pode exibir mensagens de sucesso, erro ou informações para o usuário;
- p:dialog: define um diálogo/modal do PrimeFaces que pode ser usado para exibir um formulário de cadastro de produto;

- p:outputPanel: define um painel de saída do PrimeFaces usado para agrupar e exibir conteúdo dinâmico;
- f:facet: define uma faceta do JSF que pode ser usada para adicionar conteúdo em um componente;
- action: define a ação a ser executada quando o botão é clicado;
- value: define o valor de um componente de entrada, como o texto digitado em um campo de entrada de texto.