Teste técnico Spring Boot

Projeto: API de Gestão de Produtos e Pedidos

Descrição Geral

Desenvolva uma API REST em **Spring Boot** para gerenciar produtos e pedidos de uma loja virtual. A API deve permitir a criação, leitura, atualização e exclusão (CRUD) de produtos e pedidos, e cada pedido deve estar vinculado a produtos específicos. A arquitetura deve seguir princípios **SOLID** e fazer uso de um banco de dados **SQL** para persistência.

Requisitos

1. **Entidades**:
    - **Produto**: Deve conter informações como ID, nome, descrição, preço e quantidade em estoque.
    - **Pedido**: Deve conter informações como ID, data do pedido e uma lista de produtos associados. Cada pedido deve calcular o total com base nos preços e quantidades dos produtos incluídos.
2. **Operações CRUD**:
    - **Produtos**: A API deve permitir criar, listar, atualizar e excluir produtos.
    - **Pedidos**: Deve ser possível criar um pedido e associar produtos a ele. A API também deve listar e excluir pedidos.
3. **Banco de Dados**:
    - Utilize um banco de dados **SQL** (MySQL, PostgreSQL ou H2 para testes) para persistir os dados de produtos e pedidos.
    - Configure a aplicação para que a conexão com o banco de dados seja feita via **Spring Data JPA**.

4. **Principais Endpoints**:
    - `/api/produtos`: para operações CRUD de produtos.
    - `/api/pedidos`: para operações CRUD de pedidos, incluindo a listagem dos produtos em cada pedido.
5. **Regras de Negócio e Validações**:
    - Não permitir que pedidos sejam criados com produtos que não tenham quantidade suficiente em estoque.
    - Atualizar o estoque de produtos ao criar um pedido (reduzir a quantidade conforme a demanda do pedido).
    - Implementar validações para que o preço e a quantidade de um produto nunca sejam valores negativos.
6. **Conceitos de Design (SOLID) - *Aplicar os que achar necessários para melhorar a qualidade do código***:
    - **S** (Single Responsibility): Cada classe deve ter uma única responsabilidade, seja para manipular produtos ou pedidos.
    - **O** (Open/Closed): A arquitetura deve permitir que novos recursos sejam adicionados (ex., um novo tipo de desconto) sem modificar o código existente.
    - **L** (Liskov Substitution): Crie uma interface para repositórios de dados, permitindo que substituições aconteçam (por exemplo, por mock em testes).
    - **I** (Interface Segregation): As interfaces de repositório e de serviços devem ser específicas para cada funcionalidade.
    - **D** (Dependency Inversion): Utilize injeção de dependência para que serviços e repositórios sejam facilmente substituídos e testáveis.

▼ Avaliação

☐ **Aderência a SOLID**: Verifique se os princípios de design foram seguidos corretamente.

Avaliação:

☐ **Implementação da API REST**: Avalie se a implementação dos endpoints está correta, incluindo retorno de status HTTP apropriados.

Avaliação:

☐ **Persistência e Conexão com Banco de Dados**: Confirme a correta implementação da persistência e das transações com o banco de dados.

Avaliação:

☐ **Boas Práticas e Limpeza de Código**: Verifique a legibilidade e a manutenção do código, além da clareza na documentação dos métodos, se houver.

Avaliação:

☐ **Testes Unitários e de Integração**: Verifique a implementação de testes básicos de unidade e integração, especialmente para as classes de serviço e repositório.

Avaliação: