Desafio Backend - Documentação

Processo Seletivo 25.2 ASCII EJ
Nome: **Kaike de Morais Carvalho**
Matrícula: **12421BCC051**

Esse arquivo contém todas minhas anotações, brainstorming e decisões de implementação durante o decorrer do processo de desenvolvimento do projeto. Envio isso com o intuito de mostrar como foi minha organização durante o desenvolvimento. Feito no Notion.

Descrição do Desafio

Desenvolvimento de uma API REST para gerenciamento de um sistema.

Pontos a serem avaliados:

- Boas práticas de arquitetura backend
- Clareza no código e organização
- Capacidade de integração com aplicações externas

Requisitos Funcionais

Linguagem escolhida para desenvolvimento da API REST: **Java com SpringBoot**

Banco de Dados Relacional escolhido: **PostgreSQL**

Dados a serem manipulados:

- **Nome de um produto**
- **Preço**
- **Categoria**

Requisitos Obrigatórios

Documentação por meio de um **README.md**

Modularização do projeto em camadas

Uso de um DBMS Relacional

Criação de modelos DTOs para exposição dos dados de forma padronizada

Diferenciais

Uso de **Swagger** para documentação da API

Boas práticas do código e organização

Testes Unitários

Uso de Object Relational Mapping (ORM) para manipulação do BD

Endpoints a serem Desenvolvidos

Listagem dos produtos → `GET /api/produtos`

Busca do produto por id → `GET /api/produtos/{id}`

Cria novo produto → `POST /api/produtos`

Atualiza produto → `PUT /api/produtos/{id}`

Remove produto → `DELETE /api/produtos/{id]`

Dependências Spring necessárias para o projeto

Spring Boot DevTools

Provê recursos pra facilitar desenvolvimento, como LiveReload

Spring Web

Dependência que possibilita o desenvolvimento de uma API RESTful em Spring, provendo annotations seguindo padrão Spring MVC

PostgreSQL

Permite integração com o banco de dados PostgreSQL automaticamente por meio de annotations

DataJPA

Permite o uso de Object-Relational Mapping para persistência dos dados no banco de dados, também por meio de annotations. Evita a necessidade de escrita de comandos SQL, através da API implementando Spring Data e Hibernate

Validation

Validação de Beans para verificação dos dados manipulados nos endpoints da API

Entity Relationship Diagram

> Pesquisar como que a dependência do Spring (jakarta.persistance) gera o UUID para o ID

Para a persistência e manipulação dos dados dentro do BD, desenhei a tabela de Produto seguindo a forma ilustrada acima.

Checklist e Cronograma de Desenvolvimento

> Estudar a utilidade de cada dependência Spring dentro do projeto

1. Montar projeto Spring com as dependências necessárias ✅
    a. Verificação das dependências no Maven✅
2. Configuração do Banco de Dados✅

    a. Configuração do application.properties✅
    b. Inserir application.properties no git.ignore pra evitar exposição do BD✅
3. Desenvolvimento das três camadas da API✅
    a. Modelo → annotations de Entity e Table✅
    b. Controller → annotations de RestController e RequestMapping com os endpoints respectivos✅
    c. Service → annotation de Service e processamento de cada endpoint✅
    d. Repository → annotation de Repository e extende o JpaRepository✅
4. Teste inicial no Postman sobre construção dos endpoints de POST e GET✅
5. Implementação do cálculo de valor do produto com base na quantidade e preço unitário✅
6. Construção dos restantes dos endpoints: PUT e DELETE, além do GET pelo ID
    a. PUT pra atualizar produto a partir da especificação do ID como PathVariable✅
    b. DELETE pra deletar o produto a partir da especificação do ID como PathVariable✅
    c. GET pelo ID✅
    d. GET pela categoria✅
7. Alterar atributos da tabela do BD para que os valores possam ser não-nulos✅
    a. Não coloquei UNIQUE constraint por conta da possibilidade de atualizar somente um atributo do produto através do endpoint PUT
8. Implementar Pagination✅

A partir disso, realizar a documentação da API inicial, verificando o processo de compilação e explicando da forma mais plausível possível.

Após isso, podemos expandir o projeto para:

1. Uso do Swagger

2. Testes Unitários
3. Relacionamento com múltiplas tabelas (usuários com carrinho de produtos)
4. Autenticação de usuários

Escolhas de Implementação

Brevemente aqui gostaria de explicar sobre algumas decisões de implementação.

Paginação/Pagination

Implementei paginação nos endpoints de GET, incluindo as opções de página e número de chunks a serem enviados opcionalmente como parâmetros. Isso traz eficiência ao realizar uma query no banco de dados, dando liberdade para que seja retornado apenas a quantidade necessária de informações.

Ausência de UNIQUE constraint na tabela do banco de dados

Não especifiquei a restrição de qualquer atributo dentro do BD por conta da possibilidade de registrar produtos semelhantes, diferenciando apenas por um nome de marca por exemplo. Dei liberdade para que o usuário a realizar a requisição escrever da forma que lhe convém.

UUID

Inseri UUID como forma de ID/Primary Key para a tabela do BD por questões de segurança, garantindo que todo dado cadastrado tenha seu ID único, e deixando a geração dos UUIDs por conta do próprio Spring.
