Avaliação Java / Spring: API Rest para Controle de Contatos

Descrição do Projeto

O desafio consiste em criar uma aplicação API Rest para gerenciar um sistema de cadastro de Pessoas e seus respectivos Contatos, onde cada Pessoa pode ter vários Contatos. O principal objetivo é permitir que operações CRUD (Criar, Ler, Atualizar, Deletar) sejam realizadas na estrutura de Pessoas e Contatos.

Requisitos Técnicos

- A aplicação deve ser criada utilizando Java com Spring Boot (Versão 3.2.7).
- Utilize JPA/Hibernate para persistência de dados, com banco de dados MySQL, MariaDB, PostgreSQL ou H2 como implementado em aula/exemplo: `https://github.com/eduardohen1/Java2024JP202406/tree/main/AppProdutos`
- Implemente tratamento dos dados de entrada e validações necessárias.
- A API deve ser documentada utilizando a biblioteca OpenAPI (Swagger).

Funcionalidades Necessárias

1. ***CRUD de Pessoas:***
    a. Criar Pessoa
    b. Obter Pessoa por ID
    c. Obter Pessoa por ID para mala direta
    d. Listar todas as Pessoas
    e. Atualizar Pessoa por ID
    f. Deletar Pessoa por ID
2. ***CRUD de Contatos:***
    a. Adicionar um novo Contato a uma Pessoa
    b. Obter Contato por ID
    c. Listar todos os Contatos de uma Pessoa
    d. Atualizar Contato por ID
    e. Deletar Contato por ID

Modelagem Sugerida

- ***Pessoa***: deve conter, pelo menos, os seguintes campos:
    - ID (único, não pode ser nulo)
    - Nome (não pode ser nulo)
    - Endereço (pode ser nulo)
    - CEP (pode ser nulo)
    - Cidade (pode ser nulo)
    - UF (pode ser nulo)
- ***Contato:*** deve conter, pelo menos, os seguintes campos:
    - ID (único, não pode ser nulo)
    - Tipo contato (não pode ser nulo) [inteiro] (Utilize esse conceito: 0 Telefone, 1 Celular); Pode se trabalhar com Enum também.
    - Contato (não pode ser nulo)
    - Relacionamento com a entidade Pessoa (1 pessoa para vários contatos – pesquisar relacionamento `OneToMany` e `ManyToOne`. Este relacionamento tem que ter as anotações nas duas entidades).

Endpoints **Necessários**

- ***Pessoa*****:**
    - **POST** /api/pessoas (cria uma nova Pessoa)

        - **GET** /api/pessoas/{id} (retorna os dados de uma Pessoa por ID)
        - **GET** /api/pessoas/maladireta/{id} (retorna os dados de uma Pessoa por ID para mala direta)
            - No *endpoint* de mala direta, utilizar o conceito de DTO. Este conceito cria uma classe diferente da classe Pessoa, com apenas os dados que precisamos (pesquisar!). Dê preferência para a criação de Records (Java 17+).
            - Utilizar os campos para o DTO: ID; Nome; Concatenação do Endereço, CEP, Cidade, UF
                - Exemplo:

```
{
 "ID": 1,
 "Nome": "Fulano",
 "MalaDireta": "Rua A, 1 - CEP: 11111-000 - Cidade/UF"
}
```

        - **GET** /api/pessoas (lista todas as Pessoas)
        - **PUT** /api/pessoas/{id} (atualiza uma Pessoa existente)
        - **DELETE** /api/pessoas/{id} (remove uma Pessoa por ID)
    - ***Contato***:
        - **POST** /api/contatos/ (adiciona um novo Contato a uma Pessoa)
        - **GET** /api/contatos/{id} (retorna os dados de um Contato por ID)
        - **GET** /api/contatos/pessoa/{idPessoa} (lista todos os Contatos de uma Pessoa)
        - **PUT** /api/contatos/{id} (atualiza um Contato existente)
        - **DELETE** /api/contatos/{id} (remove um Contato por ID)

Critérios de Avaliação

- Cumprimento dos requisitos funcionais.
- Organização do código, incluindo estrutura, clareza e limpeza.
- Implementação correta do relacionamento entre entidades (Pessoas e Contatos).
- Uso adequado de padrões de design e melhores práticas do Spring.
- Documentação da API com **OpenAPI** (***Swagger***), incluindo descrições claras (nas classes de *controllers*, antes da anotação de `@GetMapping` e demais, utilizar a anotação do Swagger `@Operation(summary = "explicação do endpoint aqui!")`).
- Estratégias de tratamento e validações dos dados de entrada.
- Configuração adequada do ambiente de persistência do JPA/Hibernate.
- Aplicação rodando.

Instruções de Entrega

- O código deve estar disponível no repositório público GitHub (obs.: criar o repositório em modo público).
- Incluir um README com instruções sobre como executar a aplicação, qual banco de dados utilizado, e como acessar a documentação do OpenAPI (exemplo: http://localhost:8080/swagger-ui.html).
- Enviar o endereço (link) do repositório no GitHub para entrega.

Prazo para avaliação: 10/07/2024

Bons estudos e bom trabalho!