MyCEPAPI

Arquitetura do Projeto

O projeto é uma aplicação Java com Spring Boot que implementa uma API REST para buscar CEPs e gravar os logs das consultas em um banco de dados. A arquitetura é composta pelos seguintes componentes:

1. **Controller**: Responsável por receber as requisições HTTP e retornar as respostas.
2. **Service**: Contém a lógica de negócios e faz a chamada ao serviço externo para buscar o CEP.
3. **`Repository`**: `Interface que estende JpaRepository para realizar operações de persistência no banco de dados.`
4. **Model**: Classes que representam as entidades do banco de dados.

Recursos Utilizados

- **Java 11**: Linguagem de programação utilizada.
- **Spring Boot**: Framework para simplificar a criação de aplicações Java.
- **Spring Data JPA**: Abstração para acesso a dados com JPA.
- **H2 Database**: Banco de dados em memória para desenvolvimento e testes.
- **OpenFeign**: Cliente HTTP para chamadas ao serviço externo de CEP.
- **WireMock**: Para emular o serviço externo de CEP.
- **PostgreSQL**: Banco de dados relacional utilizado em produção.
- **Docker**: Plataforma para criar, implantar e executar aplicações em contêineres.
- **Docker Compose**: Ferramenta para definir e gerenciar multi-contêineres Docker.

Princípios SOLID Aplicados

1. **Single Responsibility Principle (SRP)**: Cada classe tem uma única responsabilidade. Por exemplo, a classe `CepController` é responsável apenas por receber as requisições HTTP e retornar as respostas, enquanto a classe `CepService` é responsável pela lógica de negócios.
2. **Open/Closed Principle (OCP)**: As classes estão abertas para extensão, mas fechadas para modificação. Por exemplo, podemos adicionar novos métodos na interface `CepService` sem modificar as classes existentes.
3. **Liskov Substitution Principle (LSP)**: As subclasses devem ser substituíveis por suas superclasses. A interface `CepService` pode ser implementada por diferentes classes sem alterar o comportamento esperado.

4. **Interface Segregation Principle (ISP)**: Interfaces específicas são criadas para diferentes funcionalidades. A interface `CepService` define apenas os métodos necessários para buscar o CEP.
5. **Dependency Inversion Principle (DIP)**: As classes de alto nível não dependem de classes de baixo nível, mas de abstrações. A classe `CepController` depende da interface `CepService`, e não de uma implementação concreta.