Projeto: Tech Challenge Fase 1

Equipe: Richard de Oliveira Lopes (rm360801) e Guilherme de Oliveira Vicente (rm360802)

1. Introdução

Descrição do problema

Um grupo de restaurantes da região decidiu se unir para desenvolver, com a ajuda de estudantes, um sistema de gestão compartilhada, visando reduzir custos com soluções individuais. O sistema permitirá que clientes escolham restaurantes com base na comida, façam pedidos, deixem avaliações e acessem informações online. Devido a limitações financeiras, o sistema será desenvolvido em fases, garantindo uma implementação gradual, com ajustes contínuos conforme o uso e o feedback dos usuários.

Objetivo do projeto

Desenvolver um backend robusto utilizando Spring Boot para gerenciar usuários e atender aos requisitos definidos.

---

2. Arquitetura do Sistema

O projeto segue uma arquitetura em camadas, utilizando o framework Spring Boot para facilitar o desenvolvimento de aplicações Java. A estrutura do sistema está organizada da seguinte forma:

Camadas do Sistema

Controller: Responsável por receber as requisições HTTP, processar os dados de entrada e delegar as operações para a camada de serviço. A controller é acoplada a camada de adapters do projeto.

Service: Contém a lógica de negócios da aplicação. A classe DineWiseService é um exemplo dessa camada, onde são realizadas operações como login, criação, atualização e exclusão de usuários. Essa camada interage diretamente com a camada de repositório.

Repository: Responsável pela interação com o banco de dados. A classe UserRepositoryImpl implementa as operações de persistência, como inserção, atualização e exclusão de registros nas tabelas users e addresses. É utilizada a biblioteca spring-boot-starter-data-jdbc para simplificar o acesso ao banco de dados.

Banco de Dados

O banco de dados escolhido é o PostgreSQL. As tabelas users e addresses são utilizadas para armazenar informações de usuários e seus respectivos endereços.

Padrões de Resposta

A camada de serviço utiliza a classe ResponseEntity para padronizar as respostas HTTP, retornando mensagens e códigos de status apropriados.

---

3. Descrição dos Endpoints da API

Tabela de Endpoints

Endpoint	Método	Descrição
/api/v1/dinewise/login	POST	Login do usuário.
/api/v1/dinewise/user	POST	Criação do usuário.
/api/v1/dinewise/user/{id}	PUT	Atualiza dados do usuário, como senha, endereço, e-mail e outros dados cadastrais.
/api/v1/dinewise/user/{id}	DELETE	Deleta o usuário e o endereço do mesmo.

Exemplos de requisição e resposta

Descreva aqui exemplos de requisições e possíveis respostas.

---

4. Configuração do Projeto

Configuração do Docker Compose

Este arquivo docker-compose.yml define um ambiente de contêineres para uma aplicação e um banco de dados PostgreSQL.

A seção services define os contêineres que serão executados. Neste caso, há dois serviços: app e db.

Explicação do app:

- context: Define o diretório atual como o contexto de build.

- dockerfile: Dockerfile: Especifica que o Dockerfile no diretório atual será usado para construir a imagem do contêiner.

- ports: Mapeia a porta 8080 do contêiner para a porta 8080 do host. Isso permite acessar a aplicação no navegador ou via API em http://localhost:8080.

environment:

Define variáveis de ambiente que serão usadas pela aplicação. Estas variáveis configuram a conexão com o banco de dados PostgreSQL:

SPRING_DATASOURCE_URL: URL de conexão com o banco de dados, apontando para o serviço db na porta 5432.

SPRING_DATASOURCE_USERNAME e SPRING_DATASOURCE_PASSWORD: Credenciais para acessar o banco.

- depends_on: Define que o serviço app depende do serviço db. Isso garante que o banco de dados seja iniciado antes da aplicação.

Explicação do db:

- image: Usa a imagem oficial do PostgreSQL na versão 17-alpine, que é uma versão leve baseada no Alpine Linux.

- container_name: Nomeia o contêiner como postgres_db para facilitar a identificação.

- environment: Configura as variáveis de ambiente para o PostgreSQL:

POSTGRES_DB: Nome do banco de dados que será criado automaticamente.

POSTGRES_USER e POSTGRES_PASSWORD: Credenciais para acessar o banco.

- ports: Mapeia a porta 5432 do contêiner para a porta 5432 do host, permitindo acesso ao banco de dados localmente.

- volumes: Monta um volume chamado postgres_data no diretório /var/lib/postgresql/data dentro do contêiner. Isso garante persistência dos dados do banco, mesmo que o contêiner seja reiniciado.

5. Collections para Teste

Link para a Collection do Postman

https://github.com/guivcnt/tech_challenge_fiap1/blob/main/Fase%201.postman_collection.json

---

6. Repositório do Código

URL do Repositório

https://github.com/guivcnt/tech_challenge_fiap1
