

# Projeto: Tech Challenge Fase 2

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

## 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.

**Domain:** Responsável por ser o domínio do app, contendo as classes imutáveis (records) e independentes do framework que estamos usando.

**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.

## 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

#### # API Endpoints - DineWise

##### ## User Types

Método	Endpoint	Descrição
POST	/api/v1/dinewise/userType	Cria um novo tipo de usuário
GET	/api/v1/dinewise/userType	Lista todos os tipos de usuário
PUT	/api/v1/dinewise/userType/{id}	Atualiza um tipo de usuário pelo ID
DELETE	/api/v1/dinewise/userType/{id}	Remove um tipo de usuário pelo ID

##### ## Users

Método	Endpoint	Descrição
POST	/api/v1/dinewise/user	Cria um novo usuário
POST	/api/v1/dinewise/user/login	Realiza login de usuário
GET	/api/v1/dinewise/user	Lista todos os usuários
GET	/api/v1/dinewise/user/id/{userId}	Busca usuário pelo ID
GET	/api/v1/dinewise/user/login/{userLogin}	Busca usuário pelo login
PUT	/api/v1/dinewise/user/{userId}	Atualiza dados do usuário pelo ID
PUT	/api/v1/dinewise/user/{userId}/password	Atualiza senha do usuário pelo ID
PUT	/api/v1/dinewise/user/{userId}/type	Atualiza tipo do usuário pelo ID
DELETE	/api/v1/dinewise/user/{userId}	Remove usuário pelo ID

##### ## Menu

Método	Endpoint	Descrição
POST	/api/v1/dinewise/menu	Cria um novo menu
GET	/api/v1/dinewise/menu	Lista todos os menus
GET	/api/v1/dinewise/menu/{id}	Busca menu pelo ID
PUT	/api/v1/dinewise/menu/{id}	Atualiza menu pelo ID
DELETE	/api/v1/dinewise/menu/{id}	Remove menu pelo ID

##### ## Restaurant

Método	Endpoint	Descrição
POST	/api/v1/dinewise/restaurant	Cria um novo restaurante

```
| GET | `/api/v1/dinewise/restaurant` | Lista todos os restaurantes |  
| GET | `/api/v1/dinewise/restaurant/{id}` | Busca restaurante pelo ID |  
| PUT | `/api/v1/dinewise/restaurant/{id}` | Atualiza restaurante pelo ID |  
| DELETE | `/api/v1/dinewise/restaurant/{id}` | Remove restaurante |
```

---

## 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. Testes Unitários

Cobertura de testes realizada, utilizando as bibliotecas de JUnit, jacoco e mockito.

Element ^	Class, %	Method, %	Line, %
✓  br.com.dinewise	95% (23/24)	98% (91/92)	99% (483/484)
>  adapters.inbound	100% (5/5)	100% (27/27)	100% (31/31)
>  application	100% (10/10)	100% (56/56)	100% (444/444)
>  domain	100% (8/8)	100% (8/8)	100% (8/8)
 Application	0% (0/1)	0% (0/1)	0% (0/1)

Testes de integração automatizados realizados

## 6. Collections para Teste

Link para a Collection do Postman

[https://github.com/guivcnt/tech\\_challenge\\_fiap1/blob/main/Tech%20Challenge.postman\\_collection.json](https://github.com/guivcnt/tech_challenge_fiap1/blob/main/Tech%20Challenge.postman_collection.json)

---

## 7. Repositório do Código

URL do Repositório

[https://github.com/guivcnt/tech\\_challenge\\_fiap1](https://github.com/guivcnt/tech_challenge_fiap1)

---