Projeto: Tech-challenge-10adjt1

Filipe Gonçalves Ferreira - Rm367737

Leandro da Silva Gonçalves - Rm367789

Lucas Santos Escolástico do Nascimento - Rm367273

Índice

Resumo

O projeto visa desenvolver um sistema de gestão compartilhado para restaurantes locais, como forma de reduzir custos e padronizar a administração dos estabelecimentos participantes. Nesta segunda fase do Tech Challenge, o foco está na expansão do backend da aplicação, utilizando Spring Boot e Spring Data JPA, com persistência em banco de dados relacional MySQL.

O sistema permite a gestão de tipos de usuários, o cadastro de restaurantes e o gerenciamento dos itens de cardápio, possibilitando operações de criação, atualização, exclusão e consulta das informações. A aplicação foi estruturada seguindo os princípios da Clean Architecture, garantindo separação de responsabilidades, maior organização do código e facilidade de manutenção.

Além disso, foram implementados testes unitários e de integração para assegurar a qualidade e confiabilidade do sistema, bem como documentação da API por meio do Swagger/OpenAPI. A execução da aplicação é realizada de forma padronizada utilizando Docker Compose, permitindo um ambiente consistente para desenvolvimento e testes.

1. Introdução Descrição do problema

O avanço da tecnologia tem desempenhado um papel fundamental na otimização dos processos de gestão no setor de alimentação, especialmente no contexto de restaurantes locais. No entanto, muitos estabelecimentos enfrentam dificuldades para adotar soluções tecnológicas completas devido ao alto custo de sistemas de gestão individuais, que exigem investimentos significativos em aquisição, personalização e manutenção.

Diante desse cenário, um grupo de restaurantes de uma mesma região decidiu se unir e contratar estudantes para o desenvolvimento de um sistema de gestão compartilhado, capaz de atender simultaneamente às necessidades de diferentes estabelecimentos. A proposta busca não apenas reduzir custos, mas também criar uma solução padronizada, eficiente e escalável, que permita aos restaurantes focarem na qualidade de seus produtos e serviços, em vez de dependerem da complexidade ou da limitação de sistemas de gestão distintos.

No contexto do Tech Challenge – Fase 02, o projeto avança de forma incremental, respeitando a estratégia de entrega por fases definida inicialmente. Essa abordagem possibilita o desenvolvimento cuidadoso de cada etapa do sistema, garantindo maior controle sobre a qualidade da solução e permitindo sua evolução contínua conforme o uso e a avaliação por parte dos restaurantes e clientes.

Descrição do problema: O principal problema identificado consiste no elevado custo dos sistemas de gestão individuais disponíveis no mercado, o que inviabiliza sua adoção por parte de muitos restaurantes de pequeno e médio porte. Essa realidade levou os estabelecimentos a buscarem uma alternativa colaborativa, baseada no desenvolvimento de um sistema único e compartilhado, capaz de atender múltiplos restaurantes de forma integrada.

Além do aspecto financeiro, observa-se a necessidade de oferecer aos clientes uma experiência mais transparente e eficiente. O sistema proposto permitirá que os clientes escolham restaurantes com base na qualidade e no tipo de comida oferecida, e não em limitações ou diferenças entre sistemas de gestão utilizados pelos estabelecimentos. Para isso, a solução deverá possibilitar a consulta de informações, o registro de avaliações e a realização de pedidos online.

Outro fator relevante é a limitação de recursos financeiros para o desenvolvimento do projeto, o que motivou a adoção de uma estratégia de entrega em fases. Essa divisão permite uma implementação gradual e controlada, garantindo que cada etapa seja desenvolvida de forma cuidadosa e eficaz. Além disso, a abordagem incremental possibilita ajustes e melhorias contínuas, com base no uso real do sistema e no feedback fornecido pelos restaurantes e pelos clientes.

Dessa forma, o desafio do Tech Challenge – Fase 02 consiste em contribuir para a evolução desse sistema compartilhado, assegurando que a solução seja robusta, bem estruturada e capaz de atender às necessidades operacionais dos restaurantes, ao mesmo tempo em que oferece funcionalidades relevantes aos clientes finais.

Objetivo do projeto: O objetivo desta fase do Tech Challenge é expandir o sistema de gestão compartilhado, incorporando funcionalidades para a gestão dos tipos de usuários, cadastro de restaurantes e gerenciamento de cardápios.

Além da ampliação funcional, esta etapa reforça a aplicação de boas práticas de desenvolvimento e de estruturação de código limpo, assegurando a organização, a manutenibilidade e a qualidade do sistema. Também são contemplados requisitos técnicos voltados à documentação da API, testes automatizados e à utilização de infraestrutura Docker, permitindo uma execução integrada e padronizada da aplicação.

1.1 Introdução a Clean Architecture

A Clean Architecture é um modelo arquitetural que tem como principal objetivo promover a separação de responsabilidades dentro de uma aplicação, tornando o sistema mais organizado, testável e fácil de manter. Essa abordagem define camadas bem delimitadas, nas quais as regras de negócio permanecem independentes de frameworks, bancos de dados ou detalhes de infraestrutura.

Nesse modelo, o núcleo da aplicação concentra as regras de negócio e os casos de uso, enquanto as camadas externas são responsáveis por aspectos como interfaces de entrada, persistência de dados e configurações técnicas. Essa organização garante que alterações em tecnologias ou frameworks não impactem diretamente o domínio central do sistema.

No contexto do Tech Challenge – Fase 02, a adoção da Clean Architecture contribui para o desenvolvimento de um backend mais robusto e escalável, facilitando a aplicação de boas práticas de desenvolvimento e permitindo uma evolução controlada do sistema. Além disso, essa abordagem favorece a escrita de testes automatizados, melhora a legibilidade do código e reforça a qualidade estrutural exigida para esta fase do projeto.

1.2 Introdução ao MySQL

O MySQL é um sistema de gerenciamento de banco de dados relacional (SGBD) amplamente utilizado em aplicações web e corporativas. Ele organiza dados em tabelas que podem ser relacionadas entre si por meio de chaves primárias e estrangeiras, permitindo consultas eficientes e consistência dos dados.

No contexto do projeto, o MySQL é utilizado para armazenar informações dos usuários, incluindo clientes e donos de restaurantes, garantindo que dados como e-mail e login sejam únicos e que todas as operações de CRUD possam ser realizadas de forma segura.

Entre suas vantagens estão a confiabilidade, performance, suporte a transações ACID e compatibilidade com diversas linguagens de programação. O MySQL também é facilmente integrável com o Spring Boot, facilitando o desenvolvimento de aplicações robustas.

Exemplo prático no projeto: tabelas 'usuario', 'cliente' e 'dono_restaurante' com relacionamentos bem definidos, permitindo consultas, inserções e atualizações consistentes.

1.3 Introdução ao Spring Boot

O Spring Boot é um framework baseado em Java que simplifica a criação de aplicações stand-alone, produtivas e prontas para produção. Ele permite configuração mínima e fornece recursos integrados para desenvolver APIs REST, conectar com bancos de dados e implementar segurança.

No projeto, o Spring Boot é responsável por expor endpoints REST para o gerenciamento dos usuários, gerenciar a camada de serviço e interagir com o banco de dados via Spring Data JPA.

Entre suas vantagens estão a configuração automática, a facilidade de integração com outras bibliotecas, documentação simplificada com Swagger e a adoção de boas práticas de desenvolvimento Java.

Exemplo prático no projeto: criação de endpoints versionados (/api/v1/usuarios) para cadastro, atualização, exclusão e login de usuários.

1.4 Introdução ao Docker

O Docker é uma plataforma de virtualização que permite criar, distribuir e executar aplicações em containers, garantindo que o ambiente de execução seja consistente em diferentes máquinas.

No projeto, o Docker é utilizado para rodar tanto o banco de dados MySQL quanto a aplicação Spring Boot, orquestrados pelo Docker Compose. Isso elimina problemas de compatibilidade de ambiente e simplifica a execução.

Entre suas vantagens estão portabilidade, isolamento de dependências, facilidade de deploy e escalabilidade.

Exemplo prático no projeto: docker-compose.yml define dois serviços (app e mysql) com redes, volumes e variáveis de ambiente configuradas.

1.5 Introdução ao Spring Data JPA

O Spring Data JPA é um módulo do ecossistema Spring que tem como objetivo simplificar o acesso e a manipulação de dados em aplicações Java que utilizam bancos de dados relacionais. Ele abstrai grande parte da complexidade envolvida no uso da Java Persistence API (JPA), permitindo que o desenvolvedor foque nas regras de negócio da aplicação em

vez de lidar diretamente com código SQL ou com implementações complexas de persistência.

Por meio do Spring Data JPA, é possível realizar operações de criação, leitura, atualização e exclusão de dados (CRUD) de forma padronizada, utilizando interfaces de repositório que reduzem significativamente a quantidade de código necessário. Além disso, o framework oferece suporte à criação de consultas personalizadas, paginação, ordenação e mapeamento de entidades, garantindo maior produtividade e organização do código.

No contexto do Tech Challenge – Fase 02, o Spring Data JPA é utilizado para integrar a aplicação desenvolvida em Spring Boot ao banco de dados relacional, assegurando a integridade dos dados, a manutenibilidade do sistema e a aplicação de boas práticas de persistência. Essa abordagem contribui para a construção de um backend robusto, escalável e alinhado às exigências de qualidade e organização propostas para esta fase do projeto.

1.6 Introdução ao Teste unitários

Os testes unitários são uma prática fundamental no desenvolvimento de software, tendo como objetivo verificar o funcionamento correto das menores unidades de código de uma aplicação, como métodos ou classes, de forma isolada. Essa abordagem permite identificar falhas de maneira antecipada, garantindo que cada componente do sistema se comporte conforme o esperado.

No desenvolvimento de aplicações com Spring Boot, os testes unitários contribuem diretamente para a qualidade e a confiabilidade do código, pois asseguram que as regras de negócio sejam validadas independentemente de outros componentes, como banco de dados ou serviços externos. Dessa forma, possíveis erros podem ser corrigidos ainda nas fases iniciais do desenvolvimento, reduzindo custos e retrabalho.

No contexto do Tech Challenge – Fase 02, a utilização de testes unitários é essencial para garantir a estabilidade das funcionalidades implementadas, além de atender aos requisitos de qualidade exigidos pelo projeto. Essa prática também favorece a manutenibilidade do sistema, permitindo evoluções futuras com maior segurança e confiança no comportamento da aplicação.

1.7 Introdução ao Postman

O Postman é uma ferramenta que permite testar e documentar APIs REST. Ele facilita a criação de requisições HTTP, visualização de respostas e automação de testes.

No projeto, o Postman é utilizado para validar os endpoints, criando collections que testam cadastro, atualização, exclusão, troca de senha, login e busca de usuários.

Entre suas vantagens estão a facilidade de criação de cenários de teste, documentação interativa e suporte a ambientes distintos.

2. Arquitetura do Sistema

O sistema desenvolvido na Fase 02 do Tech Challenge segue uma arquitetura organizada em camadas, com o objetivo de garantir separação clara de responsabilidades, facilidade de manutenção e escalabilidade da aplicação. Essa estrutura possibilita que cada camada tenha uma função bem definida dentro do sistema.

A camada Controller é responsável por expor os endpoints REST, atuando como ponto de entrada das requisições realizadas pelos clientes da aplicação. Nela são recebidas as solicitações, realizados os encaminhamentos necessários e retornadas as respostas no formato adequado.

A camada Service concentra as regras de negócio do sistema, sendo responsável pelo processamento das operações e validações necessárias antes do acesso aos dados. Essa separação permite que a lógica da aplicação permaneça centralizada e independente dos detalhes de infraestrutura.

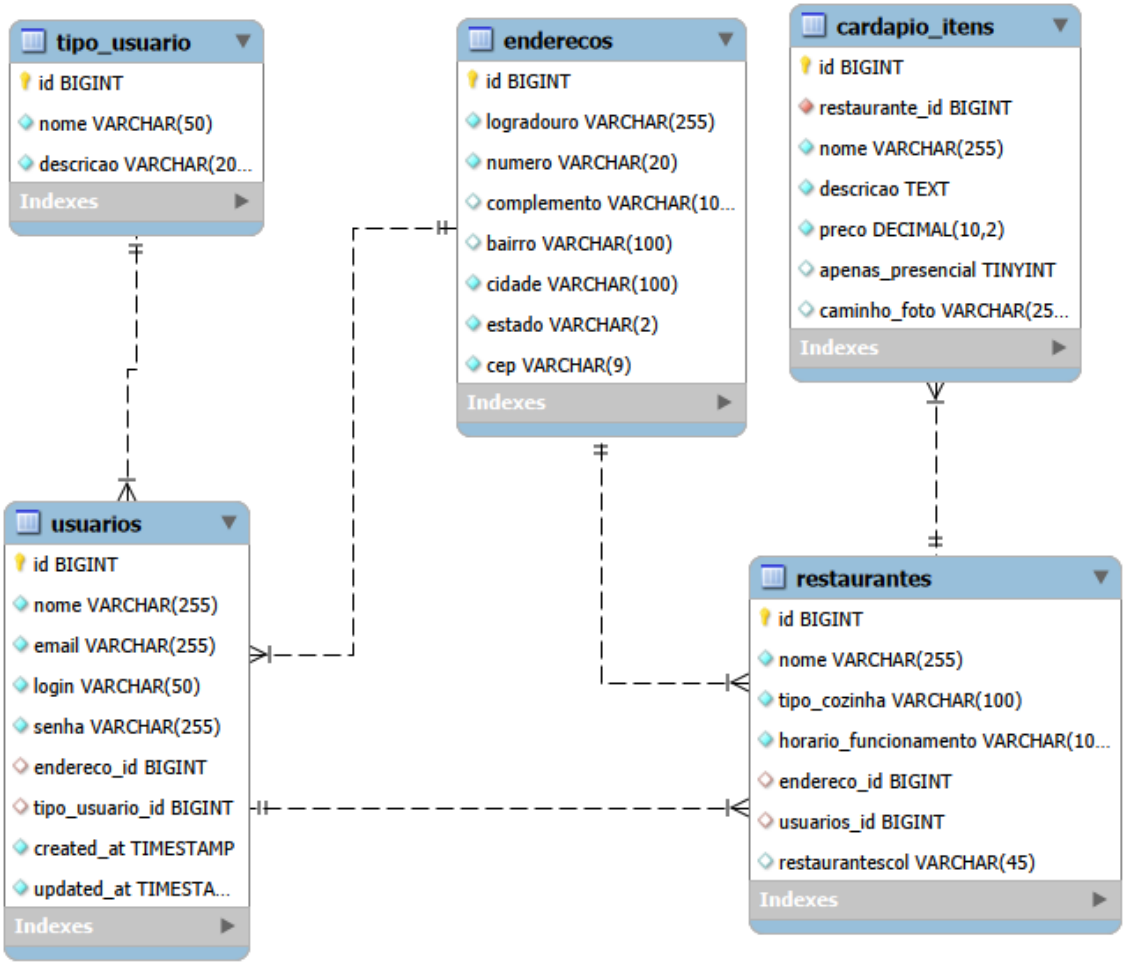
Já a camada Repository é encarregada do acesso ao banco de dados, utilizando o Spring Data JPA para realizar as operações de persistência. Essa abordagem simplifica a comunicação com o banco de dados relacional e garante maior consistência e integridade das informações armazenadas.

A aplicação foi dockerizada, permitindo sua execução em ambientes padronizados. O banco de dados MySQL é executado em um container separado, garantindo consistência, isolamento e facilidade de configuração do ambiente de desenvolvimento e testes.

Além disso, a API segue o padrão de versionamento (/api/usuário e /api/cardapio), facilitando futuras evoluções sem impacto direto nos consumidores da API.

2.1 Diagrama da Arquitetura

Diagrama simplificado:



3 Descrição dos Endpoints da API

A API de Microserviços é configurada para ser executada localmente na porta 8080. Para garantir uma organização lógica e permitir futuras expansões, foi definido o caminho base (context-path) como /api.

Todos os endpoints da aplicação utilizam este prefixo, resultando no caminho comum: http://localhost:8080/api.

O versionamento da API é implementado utilizando a estratégia Versionamento por URI, onde cada método expõe sua versão (ex: /usuário ou /cardapio) como parte do caminho, garantindo a compatibilidade retroativa e o isolamento entre as diferentes versões dos contratos de serviço.

Exemplos de requisição e resposta podem ser encontrados nas collections do Postman

Tabela de endpoints:

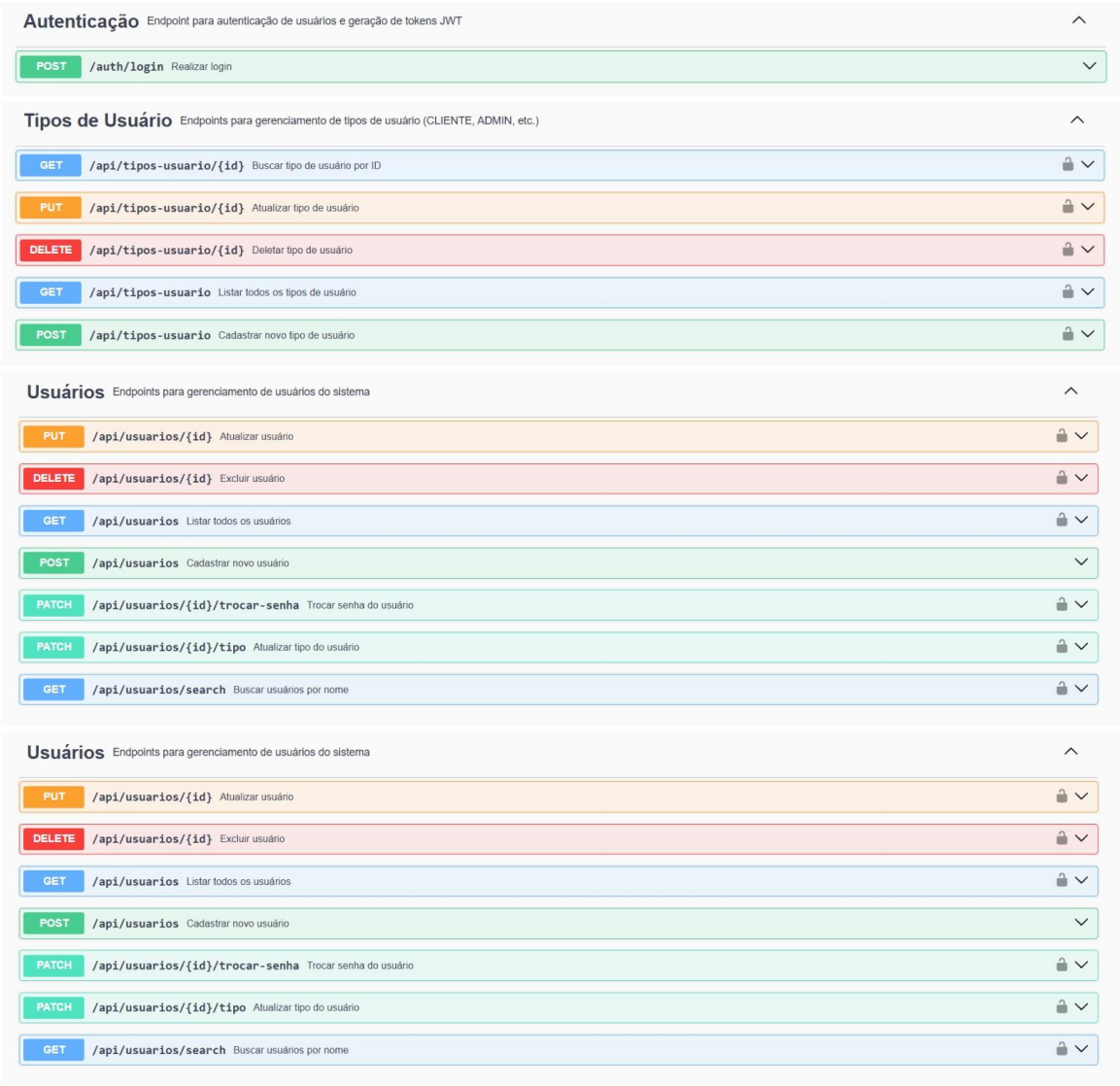
Endpoints para o gerenciamento de Autenticação		
Endpoint	Método	Descrição
/auth/login	POST	Realiza o login do usuário a partir da validação de suas credenciais, permitindo o acesso às funcionalidades da aplicação.
Endpoints para o gerenciamento de Tipos de Usuário		
Endpoint	Método	Descrição
/api/tipos-usuario/{id}	GET	Permite buscar um tipo de usuário específico a partir do seu identificador.
/api/tipos-usuario	GET	Retorna a lista de todos os tipos de usuários cadastrados no sistema.
/api/tipos-usuario	POST	Realiza o cadastro de um novo tipo de usuário.

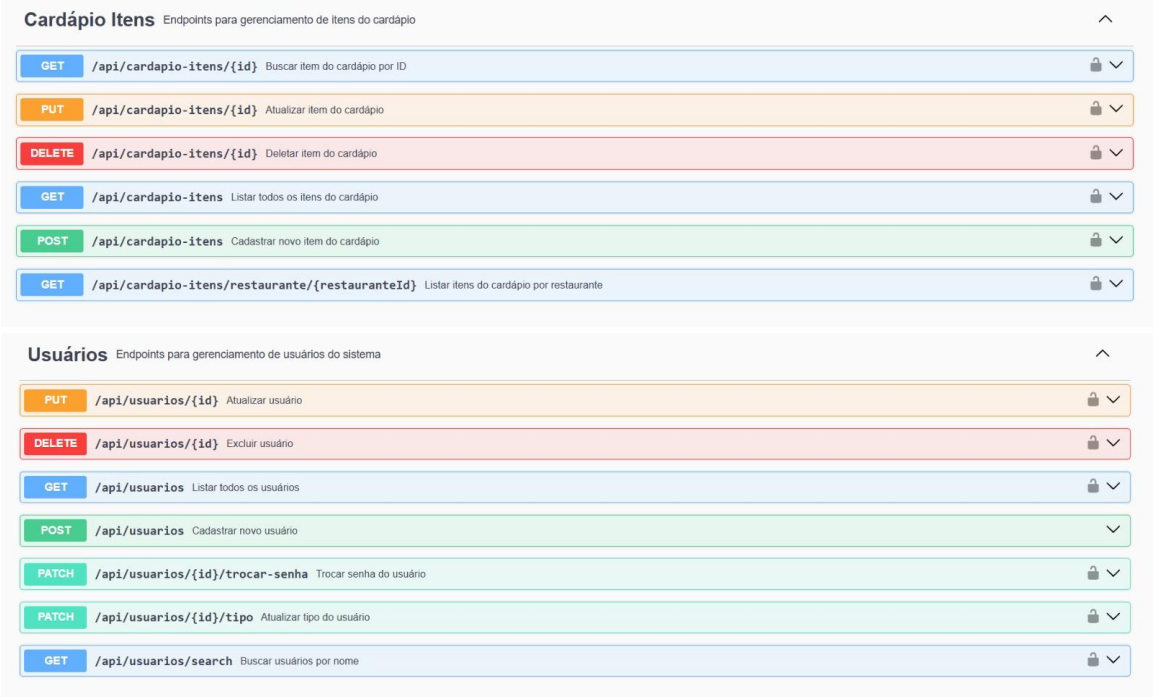
/api/tipos-usuario/{id}	PUT	Atualiza as informações de um tipo de usuário existente.
/api/tipos-usuario/{id}	DELETE	/api/tipos-usuario/{id}

Endpoints para o gerenciamento de restaurante		
Endpoint	Método	Descrição
/api/restaurantes/{id}	GET	Permite buscar um restaurante específico por meio do seu identificador.
/api/restaurantes	GET	Retorna a listagem completa dos restaurantes cadastrados no sistema.
/api/restaurantes	POST	Realiza o cadastro de um novo restaurante, associando-o a um usuário responsável.
/api/restaurantes/{id}	PUT	Atualiza as informações de um restaurante previamente cadastrado.
/api/restaurantes/{id}	DELETE	Remove um restaurante do sistema com base em seu identificador.
Endpoints para o gerenciamento de cardapio do sistema		
/api/cardapio-itens/{id}	GET	Permite buscar um item específico do cardápio a partir do seu identificador único.
/api/cardapio-itens	GET	Retorna a listagem completa dos itens de cardápio cadastrados no sistema.
/api/cardapio-itens/restaurant/{restaurantId}	GET	Lista todos os itens de cardápio associados a um restaurante específico, identificado pelo seu ID.
/api/cardapio-itens	POST	Realiza o cadastro de um novo item de cardápio, permitindo informar dados como nome, descrição, preço,

		disponibilidade e referência da imagem do prato.
/api/cardapio-itens/{id}	PUT	Atualiza as informações de um item de cardápio previamente cadastrado.
/api/cardapio-itens/{id}	DELETE	Remove um item de cardápio do sistema com base em seu identificador.

Exemplo de requisição feita no Swagger




Cardápio Itens
Endpoints para gerenciamento de itens do cardápio
GET /api/cardapio-itens/{id} Buscar item do cardápio por ID
PUT /api/cardapio-itens/{id} Atualizar item do cardápio
DELETE /api/cardapio-itens/{id} Deletar item do cardápio
GET /api/cardapio-itens Listar todos os itens do cardápio
POST /api/cardapio-itens Cadastrar novo item do cardápio
GET /api/cardapio-itens/restaurante/{restauranteId} Listar itens do cardápio por restaurante
Usuários
Endpoints para gerenciamento de usuários do sistema
PUT /api/usuarios/{id} Atualizar usuário
DELETE /api/usuarios/{id} Excluir usuário
GET /api/usuarios Listar todos os usuários
POST /api/usuarios Cadastrar novo usuário
PATCH /api/usuarios/{id}/trocar-senha Trocar senha do usuário
PATCH /api/usuarios/{id}/tipo Atualizar tipo do usuário
GET /api/usuarios/search Buscar usuários por nome

Funcionalidades Principais

Restaurantes

- Cadastrar novos restaurantes com endereço completo e tipo de cozinha
- Buscar restaurantes por ID
- Listar todos os restaurantes cadastrados
- Atualizar dados de restaurantes existentes
- Remover restaurantes do sistema

Usuários

- Cadastrar usuários com diferentes tipos (CLIENTE, ADMIN, etc.)
- Buscar usuários por nome (busca parcial)
- Atualizar dados pessoais e endereço
- Trocar senha de forma segura
- Excluir usuários

Tipos de Usuário

- Gerenciar tipos de usuário (CLIENTE, ADMIN, FUNCIONÁRIO, etc.)
- Criar, listar, atualizar e remover tipos

Códigos de Status HTTP

- **200 OK**: Requisição bem-sucedida
- **201 Created**: Recurso criado com sucesso
- **204 No Content**: Operação bem-sucedida sem retorno de dados
- **400 Bad Request**: Dados inválidos ou violação de regras de negócio
- **401 Unauthorized**: Não autenticado ou token inválido
- **404 Not Found**: Recurso não encontrado
- **500 Internal Server Error**: Erro interno do servidor

4 Cobertura de Testes Unitários

Foram implementados testes unitários abrangendo os principais serviços do sistema nesta etapa do desenvolvimento.

O objetivo foi validar as principais regras de negócio e garantir a confiabilidade das operações de criação, atualização, exclusão e consulta dos recursos.

techchallenge

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.com.fiap.infra.services		11%		0%	14	17	32	37	11	14	1	2
br.com.fiap.infra.security		58%		100%	9	19	17	51	9	14	1	3
br.com.fiap.core.usecases.usuario		95%		100%	6	80	14	204	6	49	1	19
br.com.fiap.infra.persistence.jpa.entities		92%		n/a	7	77	14	153	7	77	0	5
br.com.fiap.infra.gateways		94%		75%	11	60	5	127	2	42	0	5
br.com.fiap.core.exceptions		86%		n/a	3	21	6	42	3	21	0	18
br.com.fiap.infra.mappers		95%		100%	3	13	3	92	3	11	0	4
br.com.fiap.core.usecases.restaurante		99%		92%	1	40	1	124	0	33	0	11
br.com.fiap.core.domain		100%		91%	11	103	0	199	0	35	0	8
br.com.fiap.infra.controllers		100%		100%	0	43	0	294	0	40	0	5
br.com.fiap.core.usecases.cardapio_item		100%		100%	0	46	0	131	0	37	0	11
br.com.fiap.infra.controllers.exception		100%		n/a	0	22	0	126	0	22	0	1
br.com.fiap.infra.dto		100%		n/a	0	16	0	16	0	16	0	16
br.com.fiap.core.usecases.tipo_usuario		100%		n/a	0	25	0	57	0	25	0	11
Total	358 of 6.562	94%	27 of 292	90%	65	582	92	1.653	41	436	3	119

4.1 Frameworks e ferramentas utilizadas

JUnit 5 – para execução e estruturação dos testes unitários.

Mockito – para simulação (mock) de dependências e isolamento das camadas de serviço.

Jacoco – para medir a cobertura dos testes.

4.2 Resultado

Os testes foram executados com **sucesso**.

Obteve-se cobertura satisfatória das principais classes e métodos relacionados aos serviços.

Nenhum erro crítico foi identificado nesta etapa.

5. Autenticação da aplicação

5.1 Autenticação e Controle de Acesso

O acesso aos recursos protegidos inicia-se pelo *endpoint* de autenticação (/v1/login), onde ocorre o seguinte fluxo:

1. **Credenciais**: O usuário envia suas credenciais (e-mail e senha) via requisição POST.
2. **Autenticação**: O AuthenticationManager do Spring Security tenta autenticar o usuário, delegando a responsabilidade de criptografia e comparação de senha ao PasswordEncoder (utilizando o algoritmo bcrypt).
3. **Resposta**: O servidor retorna o JWT no corpo da resposta (TokenDTO) com o *status* 200 OK.

5.2 Tratamento de Erros e Experiência do restalrnte

Um ponto de atenção na implementação foi o tratamento de falhas de autenticação. Por padrão, o Spring Security retornaria um 403 Forbidden sem detalhes, o que é ambíguo para o cliente da API.

Em caso de credenciais inválidas (novo restaurante), o sistema captura a exceção de autenticação do Spring Security e retorna o *status* **401 Unauthorized**, acompanhado de uma mensagem clara (ex: "Nome é obrigatório.").

6. Configuração do Projeto

Docker Compose:

Define dois serviços: app (Spring Boot) e mysql (MySQL), com rede interna e volumes persistentes.

Variáveis de ambiente configuram usuário, senha e banco de dados.

Instruções para execução local.

1. Clonar o repositório.

2. Rodar 'docker-compose up -d --build'.

3. Acessar Swagger UI em http://localhost:8080/api/swagger-ui.html.

4. Testar endpoints usando Postman ou Swagger.

7. Qualidade do Código

Boas práticas utilizadas:

• Clean Architecture (Aplicação da arquitetura moderna)

• Separação de responsabilidades entre CORE e INFRA

• Uso de DTOs para transferência de dados

• Convenções do Spring Boot e Java

• Convenções do Spring Data JAP SQL

8. Collections para Teste

O Postman é utilizado para criar collections que testam todos os endpoints, incluindo:

• Atualização de dados

O arquivo da collection pode ser importado para Postman diretamente para executar os testes.

9. Repositório do Código

URL do repositório: https://github.com/lucasescol/tech-challenge-fase-02