Projeto: Tech-challenge-10adjt1

Alex Sousa Alves - Rm367425

Filipe Gonçalves Ferreira - Rm367737

Leandro da Silva Gonçalves - Rm367789

Lucas Santos Escolástico do Nascimento - Rm367273

Lucas Souza Milagres - Rm368389

Índice

Resumo

O projeto visa desenvolver um sistema de gestão compartilhado para restaurantes locais, utilizando um backend robusto em Spring Boot e banco de dados MySQL. O objetivo é otimizar a administração dos estabelecimentos e oferecer funcionalidades como cadastro, atualização, exclusão e consulta de usuários, além de recursos para clientes, como avaliações e pedidos online. O MySQL é utilizado para garantir a integridade e eficiência na manipulação dos dados, enquanto o Spring Boot facilita o desenvolvimento e integração das funcionalidades essenciais do sistema.

1. Introdução Descrição do problema

Descrição do problema: Um grupo de restaurantes locais decidiu se unir para desenvolver um sistema de gestão compartilhado, como alternativa ao alto custo de sistemas individuais. Esse sistema terá como finalidade otimizar a administração dos estabelecimentos e, ao mesmo tempo, oferecer aos clientes recursos como consulta de informações, avaliações e pedidos online.

Objetivo do projeto: Desenvolver um backend robusto utilizando Spring Boot para gerenciar usuários e atender aos requisitos definidos, incluindo cadastro, atualização, exclusão, troca de senha, validação de login, busca por nome e integração com banco de dados.

1.1 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.

1.2 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.

1.3 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.

1.4 Introdução ao SOLID

SOLID é um conjunto de cinco princípios de design orientado a objetos que visam tornar o software mais modular, flexível e de fácil manutenção.

Os cinco princípios são:

1. Single Responsibility Principle (SRP) - cada classe deve ter apenas uma responsabilidade.

2. Open/Closed Principle (OCP) - classes devem estar abertas para extensão, mas fechadas para modificação.

3. Liskov Substitution Principle (LSP) - subclasses devem poder substituir suas classes base sem quebrar a aplicação.

4. Interface Segregation Principle (ISP) - interfaces específicas são melhores do que uma única interface genérica.

5. Dependency Inversion Principle (DIP) - depender de abstrações, não de implementações concretas.

No projeto, SOLID é aplicado separando Controller, Service e Repository, usando DTOs e mantendo cada classe com responsabilidade única.

1.5 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 segue uma arquitetura em camadas (Controller, Service, Repository), permitindo separação clara de responsabilidades.

Controller: expõe os endpoints REST.

Service: contém regras de negócio.

Repository: acesso ao banco via Spring Data JPA.

A aplicação é dockerizada e o banco de dados MySQL roda em container para consistência do ambiente.

A API é versionada (/api/v1) e utiliza tratamento de erros padronizado com ProblemDetail (RFC 7807).

2.1 Diagrama da Arquitetura

Diagrama simplificado:

Diagrama de Caso de uso

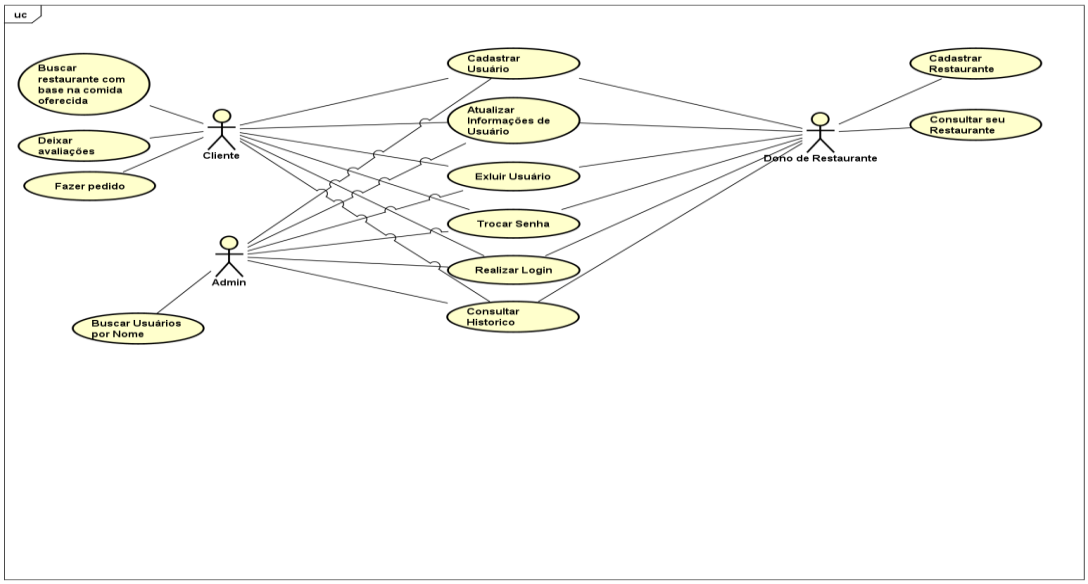
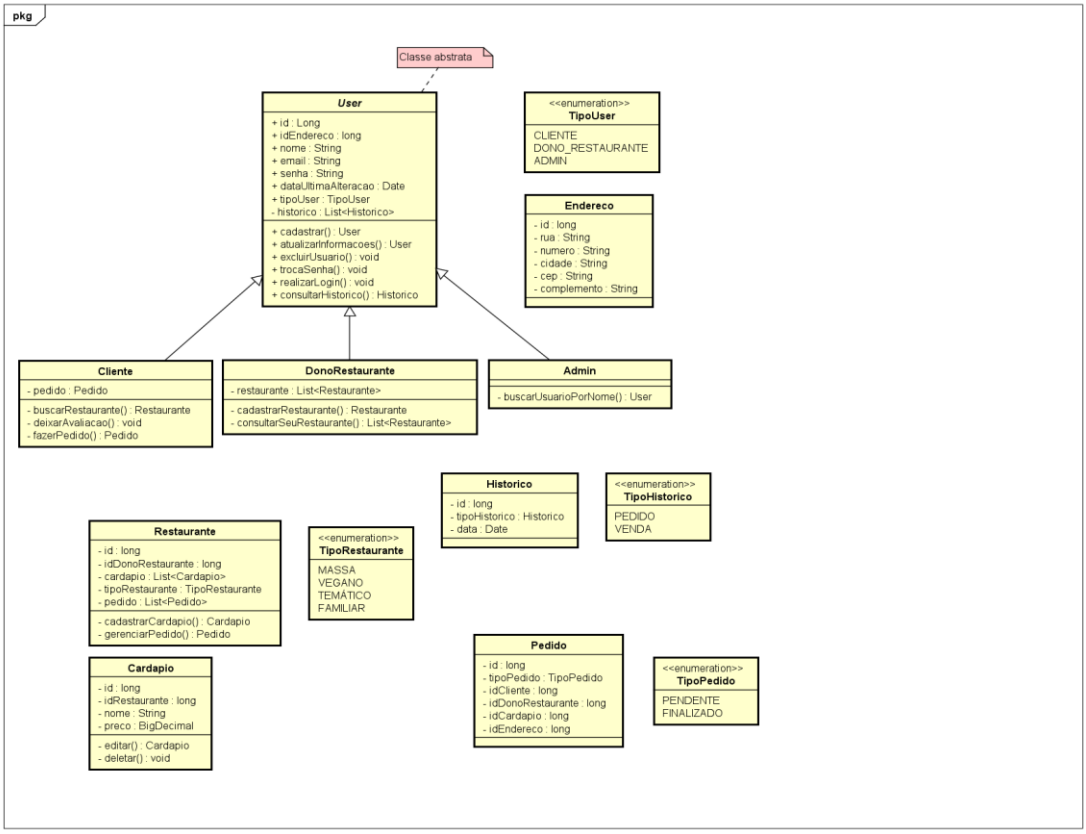


Diagrama de classes:



3. Descrição da base de dados

3.1 Módulo de Usuários e Herança (Estratégia JOINED)

O modelo centraliza as informações comuns na tabela usuario e usa tabelas filhas para definir perfis de forma flexível.

Tabela	**Propósito**	**Relacionamentos Principais**	**Chave Estrangeira (FK)**
usuario (Pai)	Centraliza dados comuns (nome, email, senha, datas).	Herança (1:1) com Cliente, Dono, Admin.	endereco (1:1)
cliente (Filha)	Perfil de Clientes.	usuario (1:1)	id -> usuario.id **(ON DELETE CASCADE)**
dono_restaurante (Filha)	Perfil de Donos de Restaurante.	usuario (1:1), restaurante (1:N)	id -> usuario.id **(ON DELETE CASCADE)**
admin (Filha)	Perfil de Administradores.	usuario (1:1)	id -> usuario.id **(ON DELETE CASCADE)**
endereco	Armazena dados geográficos (Rua, CEP, Cidade).	Usada por usuario, restaurante e pedido.	N/A

3.2 Módulo de Restaurante e Cardápio

Este módulo detalha os estabelecimentos, seus tipos e o menu disponível.

Tabela	Propósito	Chave Estrangeira (FK)
restaurante	Dados do estabelecimento.	id_dono_restaurante -> dono_restaurante.id
tipo_restaurante	Categorias possíveis (Vegano, Pizzaria, etc.).	N/A
status_restaurante	Status operacional do restaurante (ABERTO/FECHADO).	N/A
restaurante_tipo	Tabela N:N entre restaurante e tipo_restaurante.	id_restaurante, id_tipo_restaurante
restaurante_endereco	Tabela N:N para permitir múltiplos endereços/filiais.	id_restaurante, id_endereco
cardapio	Itens disponíveis no menu.	id_restaurante

3.3 Módulo de Pedidos e Histórico

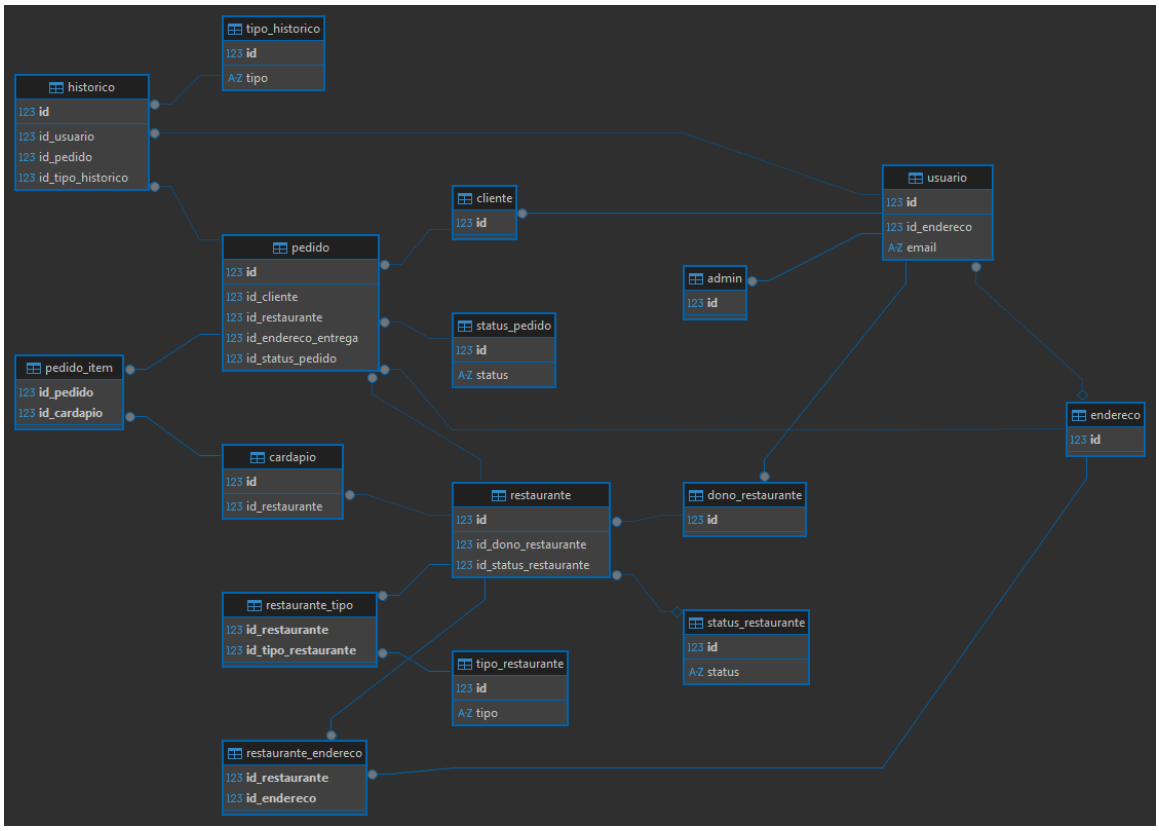
Gerencia o ciclo de vida dos pedidos, seus itens e o registro das transações.

Tabela	Propósito	Chave Estrangeira (FK)
status_pedido	Status do pedido (PENDENTE, ENTREGUE, etc.).	N/A
pedido	Registro principal da transação.	id_cliente $\rightarrow$ cliente.id
pedido_item	Detalha quais itens do cardapio compõem o pedido.	id_pedido, id_cardapio
tipo_historico	Tipos de eventos registrados (PEDIDO, VENDA).	N/A
historico	Log de eventos de usuário e pedido.	id_usuario -> usuario.id, id_pedido -> pedido.id

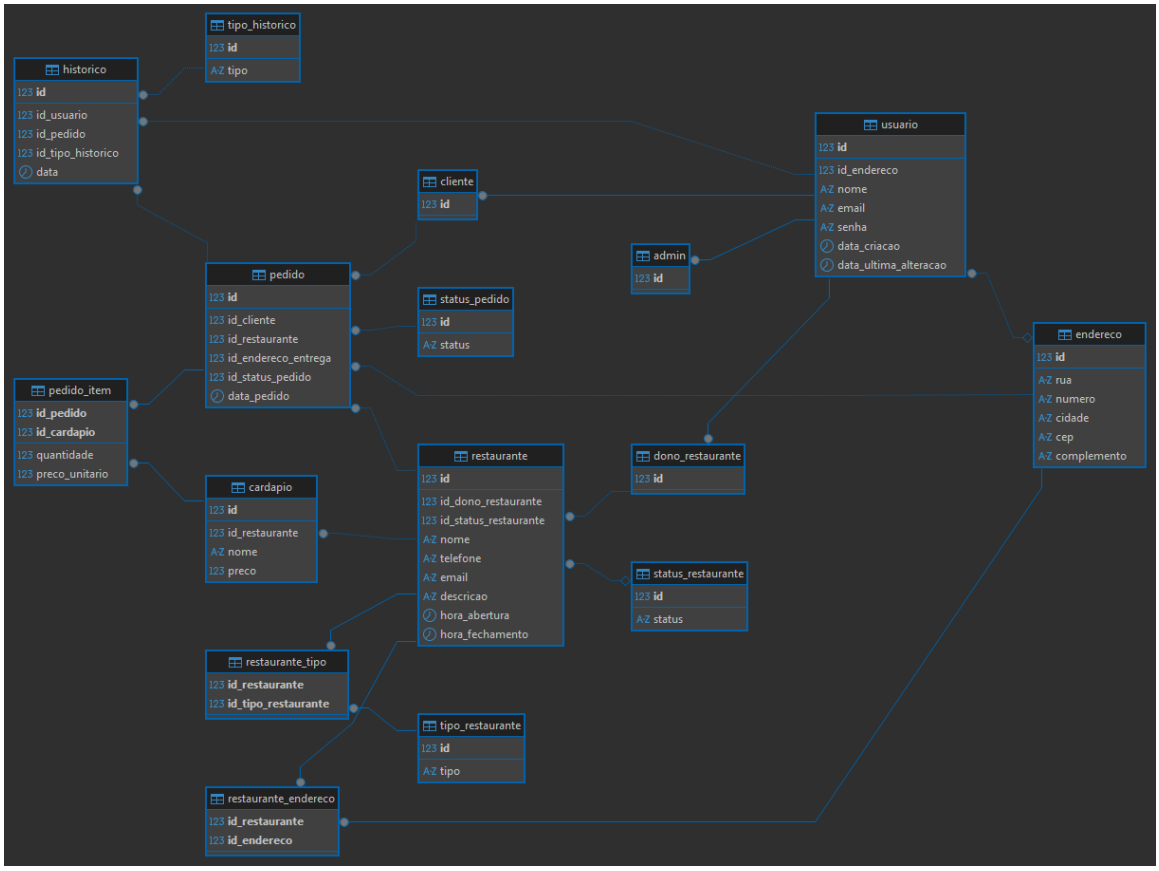
3.4 Observação de Integridade

Todas as tabelas filhas de usuario (cliente, dono_restaurante, admin) utilizam a regra ON DELETE CASCADE, garantindo que a exclusão de um registro na tabela usuario resulte na exclusão automática do registro correspondente na tabela filha, mantendo a integridade da herança.

Relacionamento das entidades (apenas as Keys)



Relacionamento das entidades (com todos os atributos)



4. 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: /v1) 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 (seção 9).

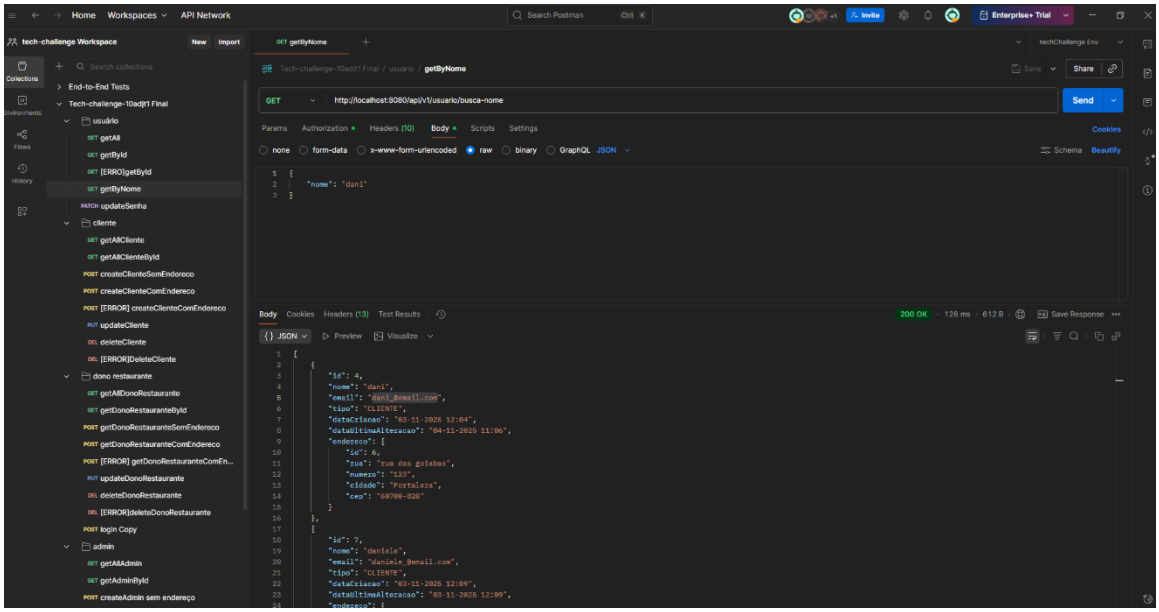
Tabela de endpoints:

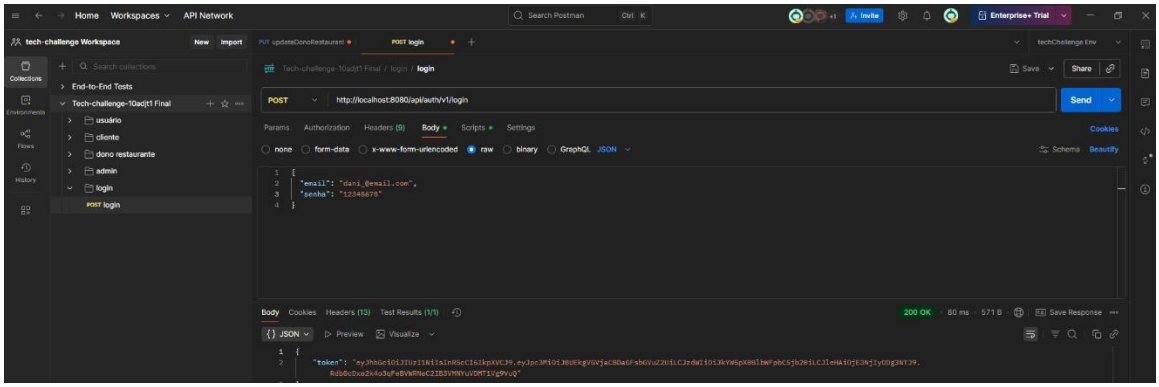
Endpoint publico		
Endpoint	Método	Descrição
/auth/v1/login	POST	Realiza o login do usuário e retorna o token JWT.
Endpoints publicos		
Endpoint	Método	Descrição
/v1/cliente	POST	Cria um novo cliente no sistema.
/V1/dono-restaurante	POST	Registra um novo dono de restaurante.
/v1/admin	POST	Cria um novo administrador.
Endpoints privados (precisa da autenticação para consumir endpoints relacionados ao Usuário)		
Endpoint	Método	Descrição
/v1/usuario	GET	Retorna a lista de todos os usuários cadastrados no sistema.
/v1/usuario/{id}	GET	Retorna os dados de um usuário específico com base no id.
/v1/usuario/888 *(erro)*	GET	Tentativa de buscar um usuário inexistente.
/v1/usuario/busca-nome	GET	Busca usuários pelo nome, geralmente com parâmetro de consulta (?nome=).
/api/v1/usuario/{id}	PATCH	Atualiza parcialmente os dados de um usuário (por exemplo, apenas o e-mail ou senha).

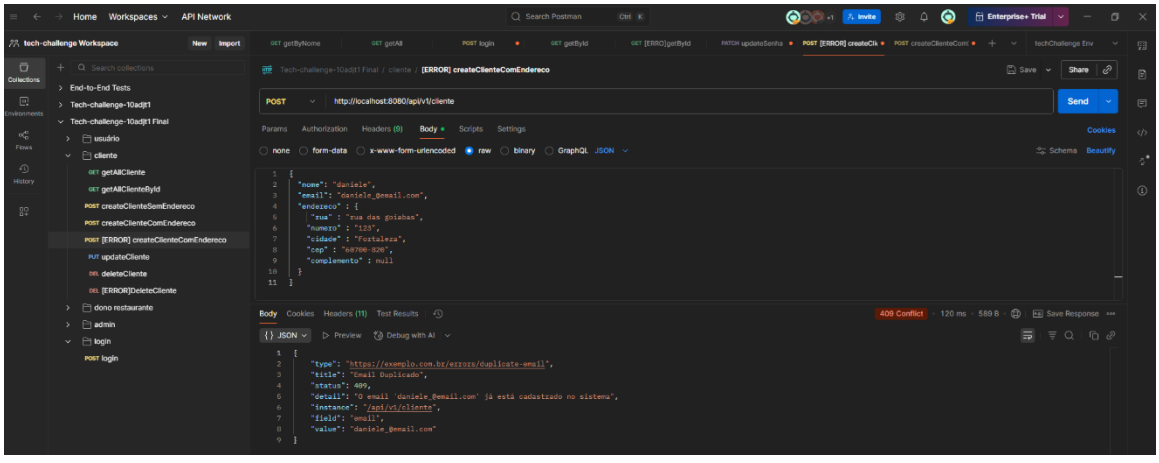
Endpoints privados (precisa da autenticação para consumir endpoints relacionados ao Cliente)		
Endpoint	Método	Descrição
/v1/cliente	GET	Retorna a lista de todos os clientes cadastrados.
/v1/cliente/{id}	GET	Retorna os dados de um cliente específico, conforme o id informado.
/v1/cliente	POST	Cria um novo cliente no Sistema relativo sem o endereço.
/v1/cliente	POST	Cria um novo cliente no Sistema absoluto com o endereço.
/v1/cliente	POST	Tentativa de criar u cliente com o mesmo e-mail, e com mesmo endereço.
/v1/cliente/{id}	PUT	Atualiza os dados de um cliente existente no sistema.
/v1/cliente/{id}	DELETE	Exclui um cliente do sistema com base no seu id.
/v1/cliente/{id}	DELETE	Tentativa de excluir um cliente do sistema com base no seu id.
Endpoints privados (precisa da autenticação para consumir endpoints relacionados ao Dono de Restaurante)		
Endpoint	Método	Descrição
/v1/dono-restaurante	GET	Retorna a lista de todos os donos de restaurante cadastrados.
/v1/dono-restaurante/{id}	GET	Retorna os dados de um dono de restaurante específico pelo id
/v1/dono-restaurante	POST	Cria um novo dono de restaurante, relativo sem endereço.
/v1/dono-restaurante	POST	Cria um novo dono de restaurante, absoluto com endereço.

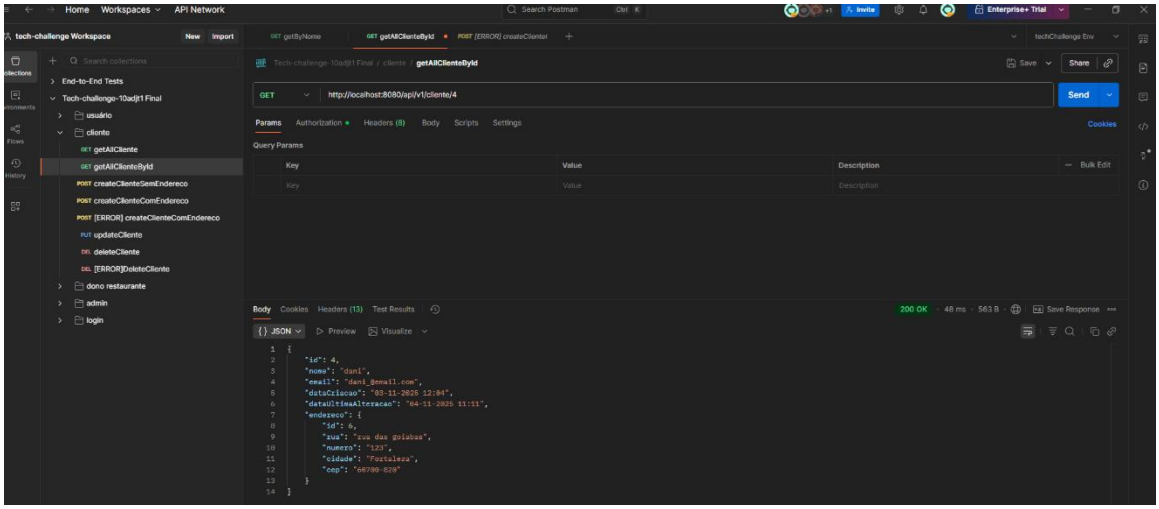
/v1/dono-restaurante	POST	Tentativa de criar um dono de restaurante com o mesmo e-mail, e com mesmo endereço.
/v1/dono-restaurante/{id}	PUT	Atualiza os dados de um dono de restaurante existente.
/v1/dono-restaurante/{id}	DELETE	Exclui um dono de restaurante com base no id
/v1/dono-restaurante/{id}	DELETE	Tentativa de excluir um dono do restaurante do sistema com base no seu id.
Endpoints privados (precisa da autenticação para consumir endpoints relacionados ao Admin)		
Endpoint	Método	Descrição
/v1/admin	GET	Retorna a lista de todos os administradores cadastrados.
/v1/admin/{id}	GET	Retorna os dados de um administrador específico pelo id
/v1/admin	POST	Cria um novo administrador, relativo sem endereço.
/v1/admin	POST	Cria um novo administrador, absoluto com endereço.
/v1/admin	POST	Tentativa de criar um admin com o mesmo e-mail, e com mesmo endereço.
/v1/admin/{id}	PUT	Atualiza os dados de um administrador existente.
/v1/admin/{id}	DELETE	Exclui um administrador com base no id
/v1/admin/{id}	DELETE	Tentativa de excluir um admin do sistema com base no seu id.

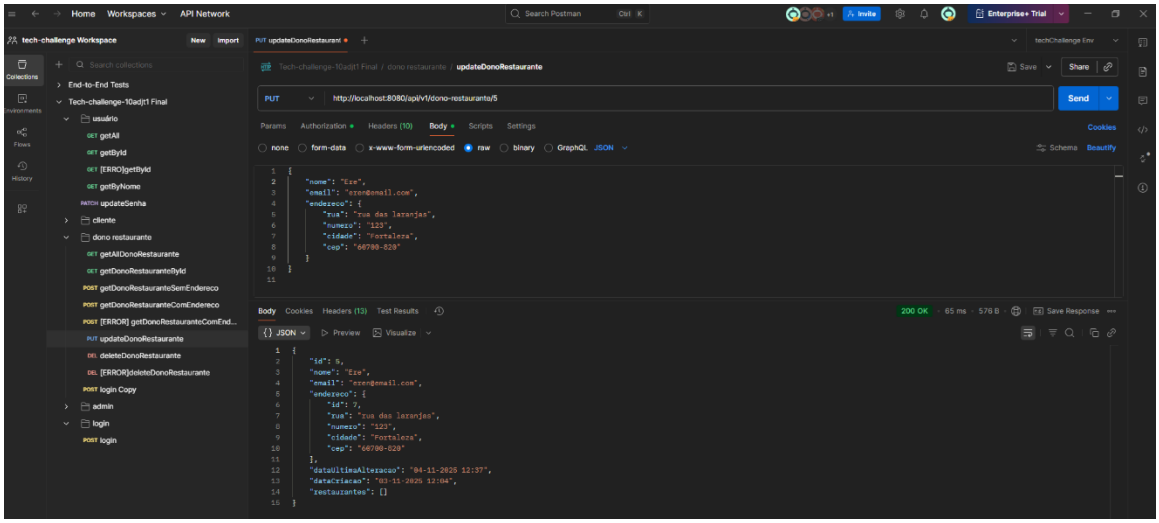
Alguns exemplos das requisições

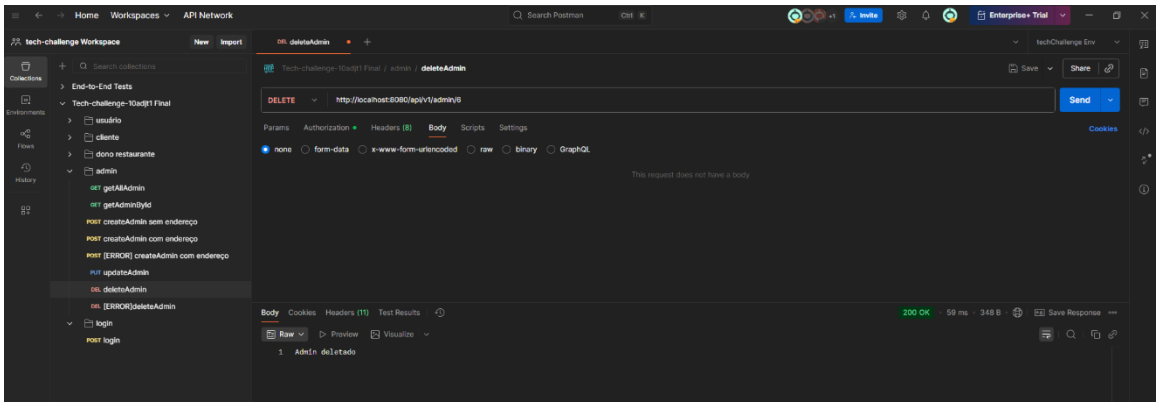




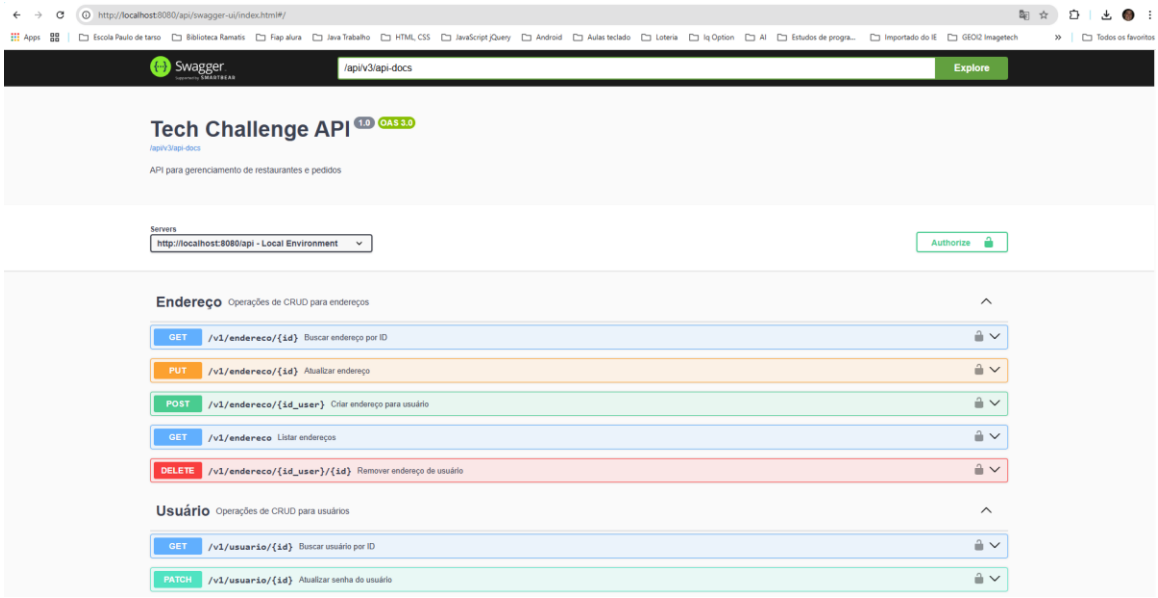



GET getAllClienteById
http://localhost:8080/api/v1/cliente/4
Send
{
"id": 4,
"nome": "dani",
"email": "dani_@email.com",
"dataCriacao": "03-11-2025 12:04",
"dataUltimaAlteracao": "04-11-2025 11:11",
"endereco": {
"id": 6,
"rua": "rua das goiabas",
"numero": "123",
"cidade": "Fortaleza",
"cep": "60700-620"
}
}


PUT updateDonoRestaurant
http://localhost:8080/api/v1/dono-restaurante/5
Send
{
"nome": "Eze",
"email": "ezer@email.com",
"endereco": {
"rua": "rua das laranjas",
"numero": "123",
"cidade": "Fortaleza",
"cep": "60700-620"
}
}
{
"id": 5,
"nome": "Eze",
"email": "ezer@email.com",
"endereco": {
"id": 7,
"rua": "rua das laranjas",
"numero": "123",
"cidade": "Fortaleza",
"cep": "60700-620"
},
"dataUltimaAlteracao": "04-11-2025 12:37",
"dataCriacao": "03-11-2025 12:04",
"restaurantes": []
}


DEL deleteAdmin
http://localhost:8080/api/v1/admin/6
Send
This request does not have a body
Admin deletado

Exemplo de requisição feita no swagger



5 Cobertura de Testes Unitários

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

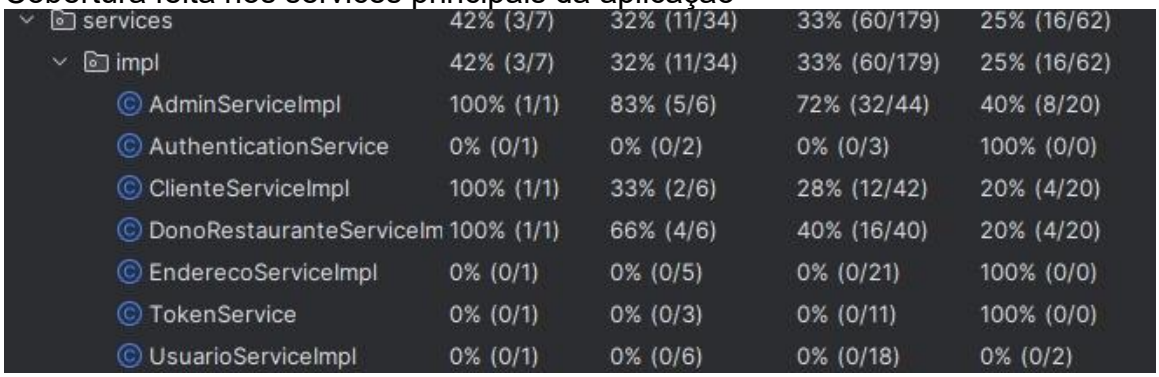
O objetivo foi validar as principais regras de negócio e garantir a confiabilidade das operações de criação, atualização, exclusão e consulta dos recursos.

5.1 Escopo dos testes

Os testes unitários contemplaram os seguintes módulos:

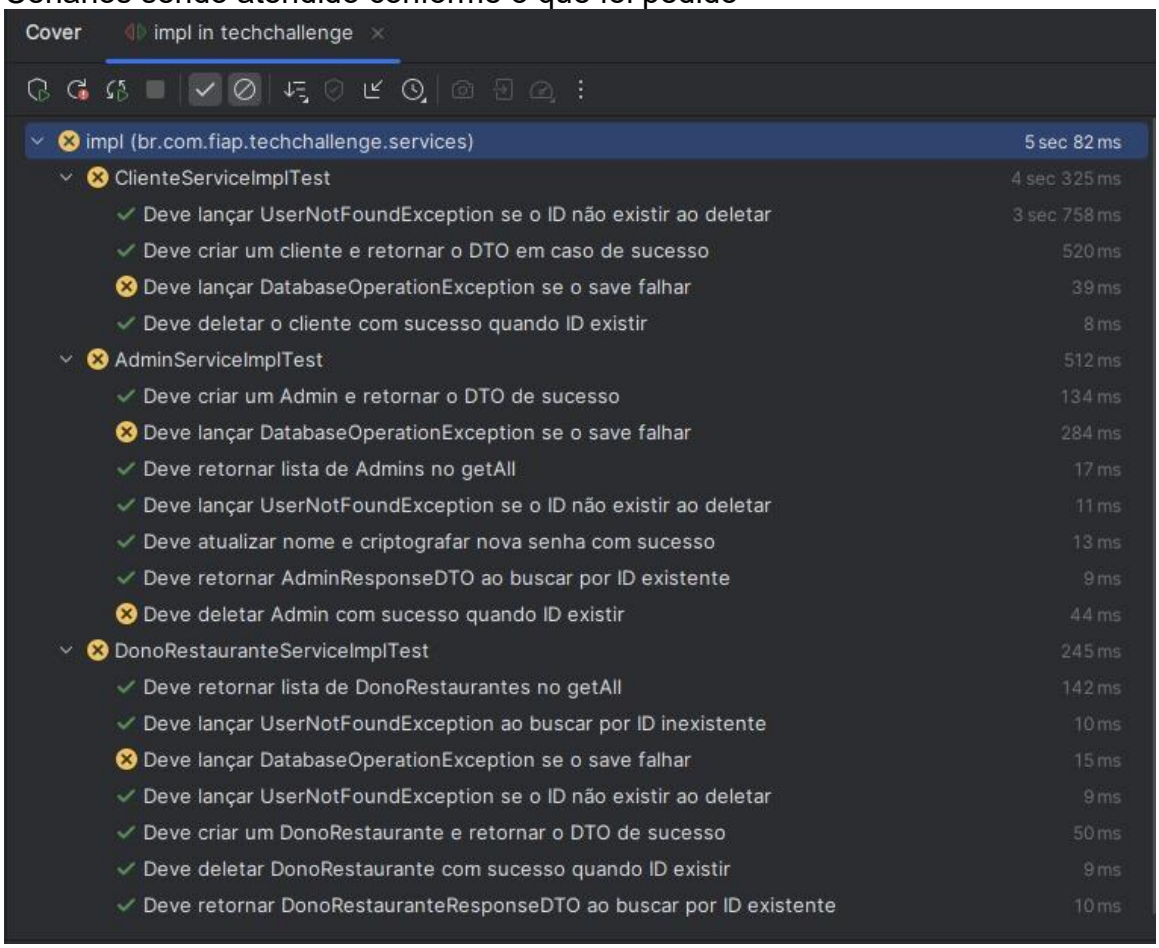
- **ClienteService** – valida operações de cadastro, atualização, busca e exclusão de clientes.
- **DonoRestauranteService** – cobre as funcionalidades de registro, atualização e listagem de donos de restaurante.
- **AdminService** – assegura o correto funcionamento das operações administrativas.

Cobertura feita nos services principais da aplicação



services	42% (3/7)	32% (11/34)	33% (60/179)	25% (16/62)
impl	42% (3/7)	32% (11/34)	33% (60/179)	25% (16/62)
AdminServiceImpl	100% (1/1)	83% (5/6)	72% (32/44)	40% (8/20)
AuthenticationService	0% (0/1)	0% (0/2)	0% (0/3)	100% (0/0)
ClienteServiceImpl	100% (1/1)	33% (2/6)	28% (12/42)	20% (4/20)
DonoRestauranteServiceIm	100% (1/1)	66% (4/6)	40% (16/40)	20% (4/20)
EnderecoServiceImpl	0% (0/1)	0% (0/5)	0% (0/21)	100% (0/0)
TokenService	0% (0/1)	0% (0/3)	0% (0/11)	100% (0/0)
UsuarioServiceImpl	0% (0/1)	0% (0/6)	0% (0/18)	0% (0/2)

Cenários sendo atendido conforme o que foi pedido



5.2 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.

5.3 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.

6. Autenticação da aplicação

6.1 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. **Geração do JWT**: Após a validação bem-sucedida, o TokenService gera um JWT. Este token é assinado criptograficamente com uma chave secreta da aplicação e contém as informações básicas de identificação do usuário (Claims).
4. **Resposta**: O servidor retorna o JWT no corpo da resposta (TokenDTO) com o *status* 200 OK.

6.2 Tratamento de Erros e Experiência do Usuário

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.

Para melhorar a experiência, implementou-se um bloco **try-catch** no método de *login* (AuthController.login).

- **Em caso de credenciais inválidas (senha ou e-mail incorretos)**, o sistema captura a exceção de autenticação do Spring Security e retorna o *status* **401 Unauthorized**, acompanhado de uma mensagem clara (ex: "Credenciais inválidas: e-mail ou senha incorretos.").

6.3 Controle de Acesso (Filtro de Segurança)

Para proteger os demais *endpoints*, um filtro de segurança (SecurityFilter) é executado antes de cada requisição. Este filtro realiza as seguintes ações:

1. **Extração do Token**: Extrai o JWT do cabeçalho Authorization (formato Bearer <Token>).
2. **Validação**: Utiliza o TokenService para validar a assinatura do token e seu tempo de expiração.
3. **Autorização**: Se o token for válido, o filtro carrega os detalhes do usuário (UserDetails) e o injeta no contexto de segurança do Spring, permitindo que a aplicação prossiga com a verificação de permissões (@PreAuthorize ou SecurityConfig).

7. 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.

8. Qualidade do Código

Boas práticas utilizadas:

• SOLID (princípios de design orientado a objetos)

• DRY (Don't Repeat Yourself)

• Separação de responsabilidades entre Controller, Service e Repository

• Uso de DTOs para transferência de dados

• Tratamento padronizado de erros com ProblemDetail

• Convenções do Spring Boot e Java

9. Collections para Teste

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

• Cadastro de usuário válido e inválido

• Atualização de dados

• Troca de senha

• Busca de usuários

• Validação de login

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

10. Repositório do Código

URL do repositório: https://github.com/gf-filipe/tech-challenge-10adjt1