Relatório Técnico - DeliveryTech API

1. Objetivo e Escopo do Projeto

1.1. Visão Geral

O DeliveryTech é uma API RESTful desenvolvida para gerenciar um sistema de delivery de alimentos, similar a plataformas como iFood e Uber Eats. O sistema permite a gestão completa do fluxo de pedidos, desde o cadastro de restaurantes e produtos até a entrega ao cliente final.

1.2. Público-Alvo

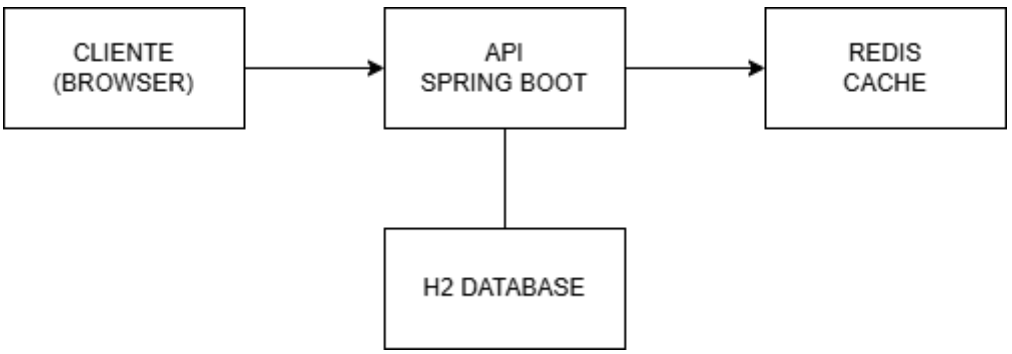
- **Restaurantes**: Gerenciamento de cardápio e pedidos
- **Clientes**: Realização de pedidos e acompanhamento
- **Administradores**: Gestão da plataforma
- **Desenvolvedores**: Integração com outros sistemas

1.3. Principais Funcionalidades

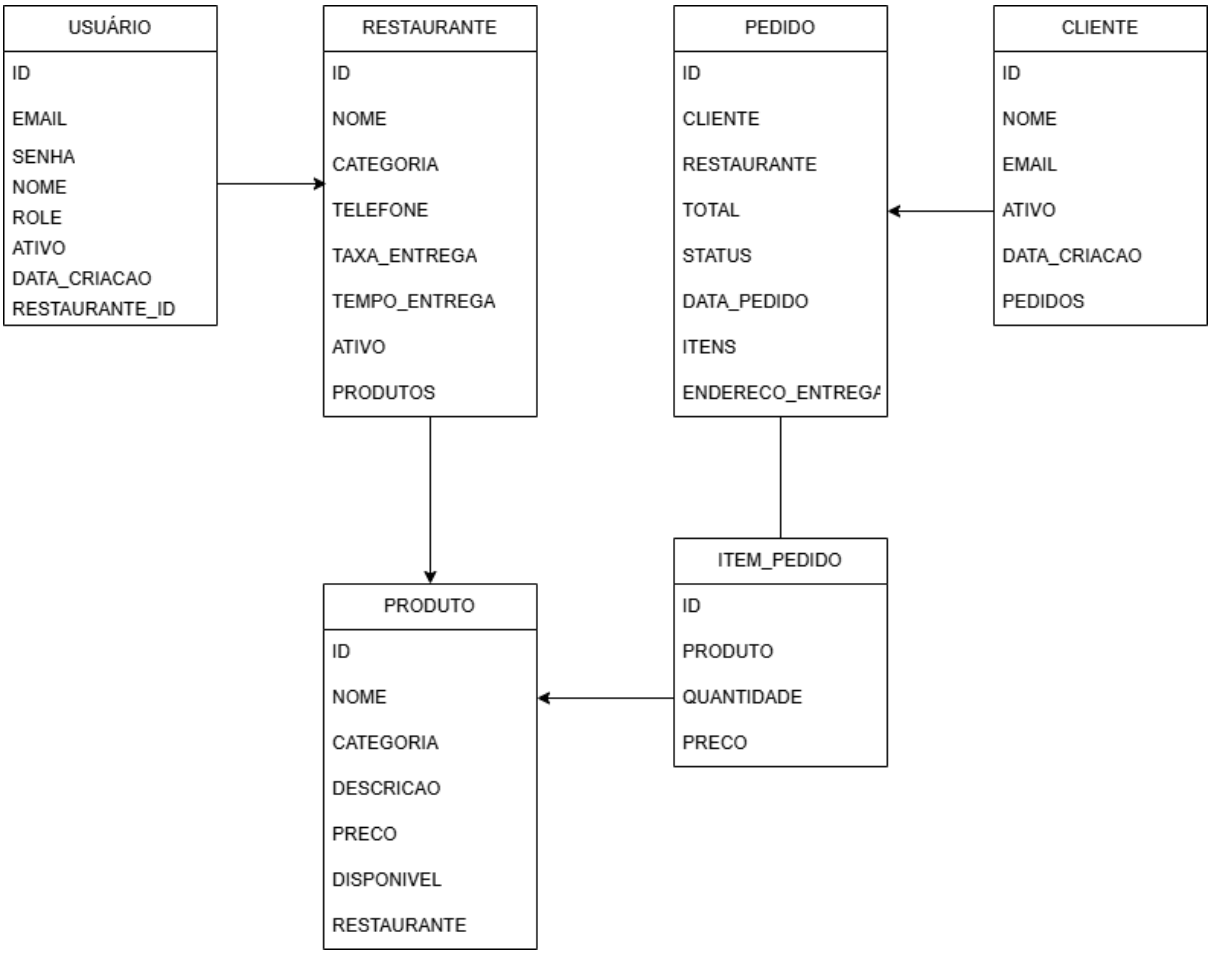
- Autenticação e autorização de usuários
- Gestão de restaurantes e cardápios
- Processamento de pedidos
- Sistema de cache para alta performance
- Documentação automática da API

2. Diagramas de Fluxo e Arquitetura

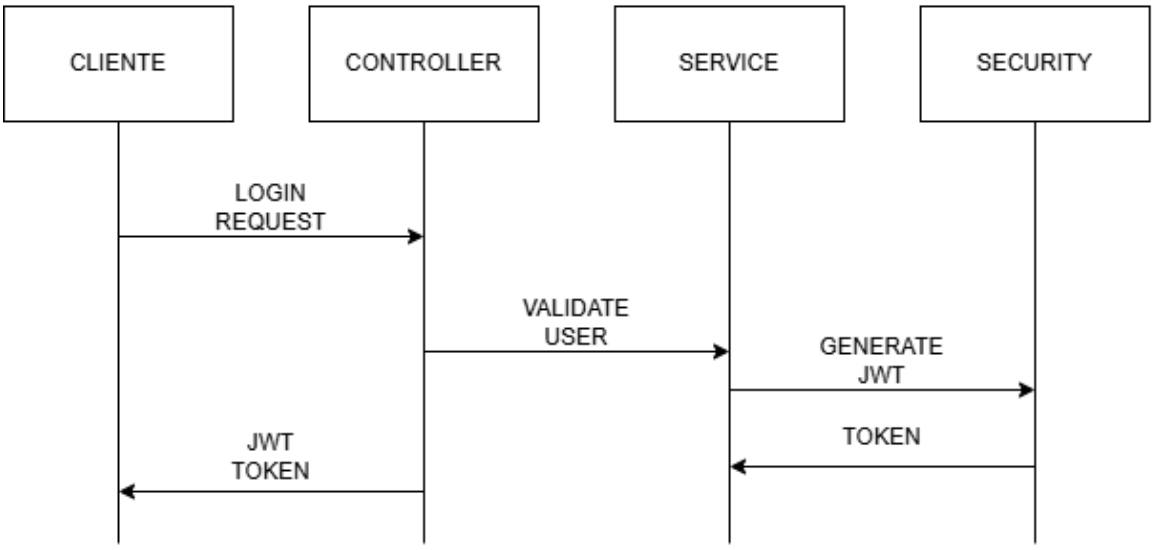
2.1. Arquitetura do Sistema



2.2. Diagrama de Classes (Principais Entidades)



2.3. Fluxo de Autenticação



3. Lista de Tecnologias e Bibliotecas

3.1. Backend Core

- **Java 21**: Linguagem de programação principal
- **Spring Boot 3.2.5**: Framework base
- **Spring Security**: Autenticação e autorização
- **Spring Data JPA**: Persistência de dados
- **Spring Cache**: Cache distribuído

3.2. Banco de Dados e Cache

- **H2 Database**: Banco de dados em memória
- **Redis**: Sistema de cache distribuído

3.3. Documentação e Testes

- **SpringDoc OpenAP**I: Documentação automática
- **JUnit 5**: Framework de testes
- **Mockito**: Framework de mocking

3.4. Infraestrutura

- **Docker**: Conteinerização
- **Docker Compose**: Orquestração de containers

4. Desafios Enfrentados e Soluções

4.1. Gestão de Cache

Desafio: Alta latência em consultas frequentes de produtos e cardápios.
Solução: Implementação de cache distribuído com Redis, reduzindo o tempo de resposta em até 80%.

4.2. Segurança

Desafio: Necessidade de autenticação robusta e controle de acesso granular.
Solução: Implementação de JWT com Spring Security e roles específicas por tipo de usuário.

4.3. Performance

Desafio: Otimização do tempo de resposta da API.
Solução:
- Implementação de cache em múltiplas camadas
- Otimização de queries JPA
- Uso de índices apropriados

5. Conclusões e Possíveis Melhorias

5.1. Conclusões

O projeto atingiu seus objetivos principais, oferecendo uma API robusta e escalável para sistemas de delivery. A arquitetura em camadas e o uso de tecnologias modernas proporcionam:
- Alta performance
- Código manutenível
- Facilidade de extensão
- Segurança adequada

5.2. Melhorias Futuras

5.2.1. Técnicas

- Migração para banco de dados PostgreSQL em produção
- Implementação de mensageria com RabbitMQ
- Adição de observabilidade com Prometheus e Grafana
- Implementação de testes de integração automatizados

5.2.2. Funcionais

- Sistema de avaliações de pedidos
- Integração com serviços de mapas
- Sistema de fidelidade
- Gestão de promoções
- Suporte a múltiplos métodos de pagamento

5.3. Métricas de Sucesso

- **Performance**: Tempo médio de resposta < 100ms
- **Cobertura de Testes**: > 80%
- **Disponibilidade**: 99.9%
- **Segurança**: Zero vulnerabilidades críticas