Relatório Técnico Estrutural

DeliveryTech API

Visão Geral do Projeto

O Delivery Tech é uma API RESTful (uma interface que permite a comunicação padronizada entre diferentes sistemas via internet) projetada para gerenciar o ecossistema completo de um aplicativo de delivery de comidas, no estilo de plataformas como iFood, Rappi e Uber Eats. O sistema gerencia todo o fluxo de valor, desde o momento em que o restaurante cadastra seu cardápio, até a entrega do pedido na casa do cliente.

Para quem estamos construindo?

O Delivery Tech é uma plataforma que pretende atender a quatro fundamentais pilares:

- **Restaurantes:** Interface para gerir o negócio (cardápios, preços e entregas);
- **Clientes:** Usuários que buscam facilidade para explorar opções, fazer e acompanhar pedidos, e a entrega.
- **Administradores:** Equipe interna que monitora a saúde, segurança e regras de negócio da plataforma.
- **Desenvolvedores:** Profissionais que consumirão a API para criar os app's mobile ou web.

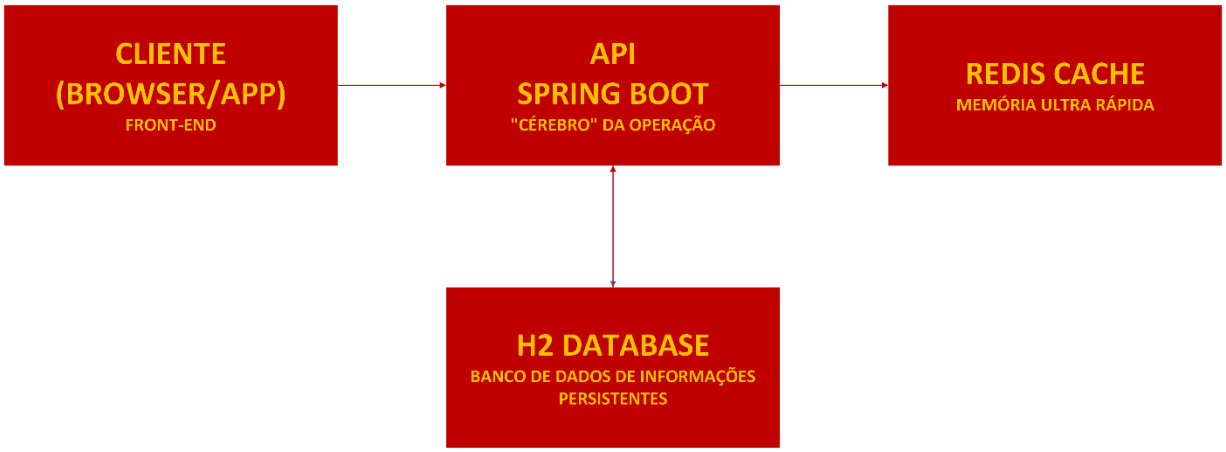
Funcionalidades

Principais funcionalidades do DeliveryTech:

- **Segurança em primeiro lugar:** Autenticação e autorização rigorosa para proteção de dados dos nossos clientes.
- **Gestão de Empresas:** Completo para restaurantes e cardápios.
- **Motor de Pedidos:** Processamento inteligente e rastreável do carrinho de compras até a entrega.
- **Alta Performance:** Sistema de cache para garantir que o aplicativo não trave em horários de pico (como um sexta-feira à noite).
- **Auto Documentação:** Rotas mapeadas automaticamente para facilitar o trabalho do front-end.

Como o sistema funciona?

A aplicação segue uma arquitetura moderna e dividida em responsabilidades claras:



- **Cliente (Browser/App):** Envia as requisições HTTP.
- **API Spring Boot:** O "cérebro" da operação, onde a lógica de negócio acontece. É aqui que seus arquivos de controle (como um ClienteController) recebem as chamadas.
- **H2 Database:** O banco de dados relacional onde as informações persistentes moram (ideal para o ambiente de desenvolvimento atual). É acessado através dos repositórios (como um ClienteRepository).
- **Redis Cache:** Uma memória ultra-rápida. Em vez de perguntar ao banco de dados "quais são os restaurantes disponíveis?" toda vez, a API guarda essa resposta no Redis temporariamente para entregá-la em milissegundos.

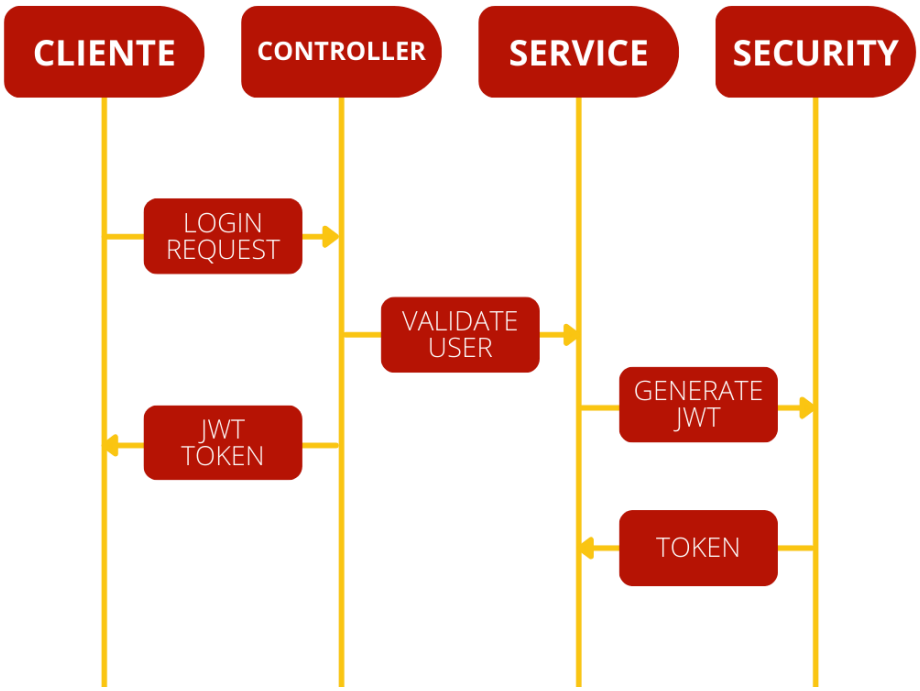
O Coração dos Dados - Modelo de Domínio

O banco de dados foi modelado para refletir a realidade de um delivery:

- **Usuário:** Pessoa responsável por um ou mais restaurantes, com autenticação no sistema, ou pessoa da administração do domínio.
- **Restaurante:** Parte onde vai conter as informações do Restaurante em questão, como telefone, taxa de entrega, produtos, se está aberto ou não, entre outras informações.
- **Produto:** Informações e descrição sobre cada produto pertencente aquele restaurante
- **Cliente:** Usuário (também autenticado no sistema) pode realizar pedidos.
- **Pedido:** Contendo todos os dados do cliente, data, itens e local para entrega.
- **Itens Pedidos:** Lista a ser encaminhada para o restaurante dos itens pedidos pelo cliente.

O Fluxo de Segurança

Para garantir que apenas pessoas autorizadas acessem os dados, implementamos um fluxo de login robusto (gerenciado por configurações como um SecurityConfig).



1. O Cliente envia suas credenciais (Login Request) para o Controller.

2. O Controller repassa os dados para o Service, que aciona a camada de Security para validar o usuário.

3. Se aprovado, o sistema gera um Token JWT (uma espécie de "crachá digital" criptografado) e o devolve ao cliente. As próximas requisições usarão esse token para comprovar a identidade do usuário.

Base Tecnológica – Stack

As ferramentas foram escolhidas visando longevidade, suporte da comunidade e escalabilidade:

- **Linguagem & Framework:** Uma combinação poderosa que oferece recursos modernos e produtividade acelerada.
    - **JavaSE-21 –** Linguagem de programação
    - **Spring Boot 3.2.5 –** Framework base
- **Dados e Performance:** Spring Data JPA simplifica a comunicação com o banco H2, enquanto o Spring Cache e o Redis garantem respostas na velocidade da luz.
    - **Spring Data JPA –** Persistência de dados
    - **H2 Database –** Banco de dados persistente
    - **Spring Cache –** Cache distribuído
    - **Redis –** Sistema de cache distribuído
- **Dados e Performance:** Para blindar a API contra acessos indevidos.
    - **Spring Security –** Autenticação e autorização
- **Qualidade e Infraestrutura:** JUnit 5 e Mockito garantem que o código funcione através de testes automatizados. Docker e Docker Compose empacotam a aplicação para que ela rode perfeitamente em qualquer máquina. SpringDoc OpenAPI gera um manual de uso interativo da API automaticamente.
    - **JUnit 5 –** Framework de testes
    - **Mockito –** Framework de mocking
    - **Docker –** Conteinerização
    - **Docker Compose –** Gestão de containers
    - **SpringDoc OpenAPI –** Documentação automática.

Problemas Reais, Soluções Inteligentes

Durante o desenvolvimento, superamos obstáculos críticos para uma plataforma de escala:

- **Desafio de Latência:** Consultar o banco de dados toda vez que um usuário abria o cardápio deixava o app lento.
    - ***Solução:*** Implementamos o Redis como um cache distribuído. Resultado: O tempo de resposta caiu em até 80%.

- **Desafio de Acesso:** Como separar o que um Restaurante pode ver do que um Cliente pode ver?
    - ***Solução:*** Uso de Tokens JWT com injeção de Roles (papéis). O sistema sabe exatamente "quem é quem" em cada requisição.

- **Desafio de Gargalo de Dados:** O sistema estava consumindo muito recurso em pesquisas complexas.
    - ***Solução:*** Otimizamos as consultas (queries JPA) e adicionamos índices no banco de dados, além de um cache em múltiplas camadas.

Onde Estamos e Para Onde Vamos

Status Atual

A fase inicial foi um sucesso. Temos uma arquitetura baseada em camadas que não apenas funciona, mas entrega alta performance, segurança e é fácil de manter ou expandir.

Atingimos métricas de excelência:

- Respostas da API em menos de 100ms.
- Mais de 80% do código coberto por testes.

- Zero vulnerabilidades críticas de segurança identificadas.
- Arquitetura preparada para 99.9% de disponibilidade.

O Próximo Nível (Roadmap)

Para preparar a DeliveryTech para o ambiente de produção real, os próximos passos incluem:

- **Evolução Técnica:** Trocar o banco em memória (H2) por um banco robusto definitivo (PostgreSQL). Implementar RabbitMQ para mensageria assíncrona (ex: avisar a cozinha sem travar a tela do usuário) e Prometheus/Grafana para monitorar a saúde dos servidores.

- **Evolução do Produto:** Adicionar recursos essenciais para engajamento, como sistema de avaliações, integração com mapas (GPS), programas de fidelidade, cupons de desconto e gateways de pagamento múltiplos.

Ideias para front-end.

Logo

Tela de Login

Cardápio

Entrega