1. Objetivo e escopo do projeto

O **DeliveryTech** é uma API RESTful desenvolvida para gerenciar o fluxo operacional de um sistema de entregas. O objetivo principal é conectar três atores centrais: **Clientes**, **Restaurantes** e **Pedidos**, garantindo a integridade dos dados e a segurança das transações.

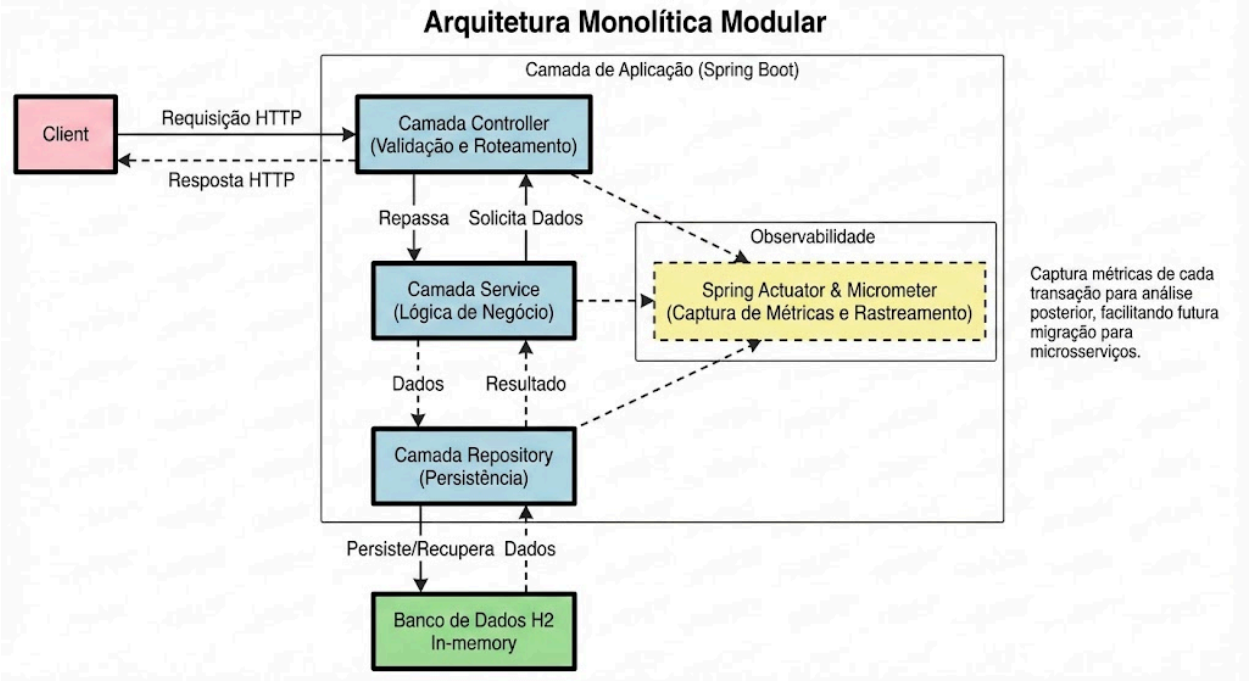
- **Escopo:**
    - Gestão de usuários com diferentes níveis de acesso (Roles).
    - Catálogo de produtos vinculado a estabelecimentos específicos.
    - Processamento de pedidos com rastreamento de status em tempo real.
    - Monitoramento ativo de performance e saúde da aplicação.

---

2. Diagrama de fluxo e Arquitetura

A arquitetura segue o padrão **Monolítico Modular**, preparada para futura migração para microsserviços graças ao uso de tecnologias de rastreamento distribuído.

- **Fluxo de Dados:** O cliente interage com a camada de controller, que valida a requisição e a repassa para a camada de service (onde reside a lógica de negócio), finalizando na persistência via repository no banco de dados **H2**.
- **Observabilidade:** Integrado ao fluxo principal, o **Spring Actuator** e o **Micrometer** capturam métricas de cada transação para análise posterior.



---

3. Telas implementadas e descrições

Embora o projeto seja focado em Backend, utilizamos interfaces de gerenciamento e documentação essenciais:

- **Swagger UI (SpringDoc):** Interface principal de interação. Permite que desenvolvedores e integradores testem cada endpoint da API (POST, GET, PUT, DELETE) sem a necessidade de um Frontend pronto.

- **H2 Database Console:** Interface de visualização do banco de dados em memória, utilizada para validar se as entidades (Cliente, Pedido) estão sendo persistidas corretamente conforme o modelo JPA.

- **Actuator Health/Metrics:** Endpoints JSON que fornecem o status "vivo" da aplicação e estatísticas de uso de memória e CPU.

---

4. Lista de tecnologias e bibliotecas

O projeto utiliza o estado da arte do ecossistema Java:

Categoria	**Tecnologias Utilizadas**
Core Framework	Spring Boot 3.2.5
Segurança	Spring Security & JWT
Persistência	Hibernate / Spring Data JPA
Banco de Dados	H2 Database (In-memory)
Observabilidade	Actuator, Micrometer, Zipkin
Documentação	SpringDoc OpenAPI (Swagger)

Containerização	Docker & Docker Compose

---

5. Desafios enfrentados e soluções aplicadas

1. **Conflitos de Versão no Maven:** Identificamos que a declaração manual de versões no pom.xml causava erros de carregamento no Actuator.
    - *Solução:* Removemos as versões explícitas, permitindo que o spring-boot-starter-parent gerenciasse as dependências de forma harmônica.

2. **Repositórios Git Aninhados:** O erro "fatal" ao tentar subir o código ocorreu por tentar inicializar o Git em uma pasta superior à do projeto.
    - *Solução:* Resetamos a configuração do Git e inicializamos o repositório exatamente na raiz do projeto (deliverytech), garantindo uma estrutura limpa no GitHub.

3. **Ambiente de Desenvolvimento:** Configuração do VS Code para reconhecer o classpath corretamente após mudanças estruturais.
    - *Solução:* Limpeza do repositório local .m2 e uso do comando mvn clean install -U.

---

6. Conclusões e possíveis melhorias

O projeto atingiu um nível de maturidade técnica que o qualifica para um ambiente de "Pré-Produção". A estrutura é organizada, testável e monitorada.

Sugestões para o futuro:

- **Cache com Redis:** Implementar cache na busca de produtos para reduzir a carga no banco de dados principal.

- **Integração com Gateway de Pagamento:** Adicionar uma camada de serviço para processamento de cartões e PIX.

- **Migração para PostgreSQL:** Substituir o H2 por um banco persistente em disco para suportar dados reais em produção.

- **Módulo de logística (Entregadores):** Criação de uma nova entidade (`Entregador`) e uma nova Role de segurança (`ROLE_COURIER`). Isso incluirá o mapeamento de dados pessoais, veículo utilizado e a implementação de uma máquina de estados no `Pedido` para incluir o status "EM_ROTA", vinculando o pedido ao entregador **responsável.**

- **Sistema de avaliação e Feedback:** Implementação de uma entidade de `Avaliacao` vinculando Cliente, Pedido, Restaurante e Entregador. Para garantir a performance e evitar consultas pesadas (queries de agregação) no banco de dados principal, a consolidação das notas médias dos restaurantes poderá ser armazenada em cache (Redis).

- **Consumo via Front-end (Web/Mobile):** Construção de interfaces para Restaurantes (Dashboards Web) e Clientes/Entregadores (Apps Mobile), incluindo a devida configuração de CORS no Spring Security para comunicação entre as pontas.