Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas - AMS

Beatriz Alamino

Cauan Ortiz

Enrico Meira

Filipe Fogaça

Marcio Torres

Documentação Técnica de Desenvolvimento de Software

Binance

Sorocaba
Dezembro - 2025

Documentação Técnica de Desenvolvimento de Software

Binance

Beatriz Alamino

Cauan Ortiz

Enrico Meira

Filipe Fogaça

Marcio Torres

Projeto final apresentado à Faculdade de Tecnologia de Sorocaba, como parte dos pré-requisitos para finalização de disciplina Programação Multiplataforma do curso de Análise e Desenvolvimento de Sistemas do programa de Articulação do Ensino Médio e Superior do Centro Paula Souza.

Orientador: Prof. André Cassulino

Sorocaba

Dezembro - 2025

Índice

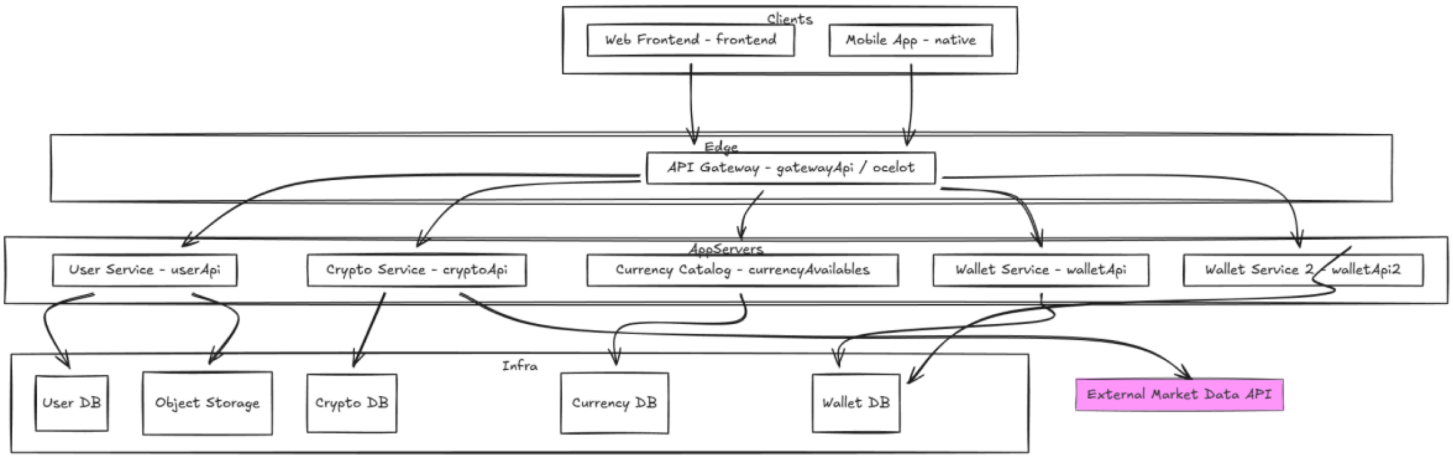
1. Diagrama de Arquitetura Geral do Sistema 5
Descrição da Arquitetura Geral - 5
2. Diagramas de Classes (UML) 6
Descrição WalletAPI - 6
Descrição UserAPI - 7
Descrição Cerrency - 7
3. DER — Diagrama Entidade-Relacionamento 8
Descrição do Modelo de Dados - 8
4. Diagramas de Sequência 9
Descrição Login - 9
Descrição Depósito - 9
Descrição Trade - 10
Descrição Depósito via Chatbot - 10
5. Fluxo de Comunicação entre Serviços 11
6. Descrição textual dos componentes e tecnologias 12
6.1. Visão Geral da Aplicação 12
6.2. Componentes da Arquitetura 12
6.2.1 UserAPI — Serviço de Usuários 12
6.2.2 WalletAPI — Serviço de Carteiras e Transações 12
6.2.3 CurrencyAPI — Serviço de Preços e Ativos 13
6.2.4 ChatbotAPI — Serviço de Chat Automatizado 13
6.2.5 GatewayAPI — API Gateway da Plataforma 14
6.3. Frontend e Aplicativo Mobile 14
6.3.1 Frontend Web (Next.js + TypeScript) 14
6.3.2 Aplicativo Mobile (React Native + Expo) 15
6.4. Componentes de Mensageria (RabbitMQ) 15
6.5. Segurança e Autenticação 15

6.6. Clean Architecture dentro dos Microserviços 16
6.7. Banco de Dados — SQLite 16
6.8. Controle de Versão — GitHub + GitFlow 16
6.9. Tecnologias Principais e Justificativa de Uso 17

1. Diagrama de Arquitetura Geral do Sistema

Descrição da Arquitetura Geral -

A arquitetura do sistema de corretora de criptomoedas é baseada em um conjunto de microserviços independentes, cada um responsável por uma parte específica do domínio.
A comunicação entre os clientes (Frontend web e aplicativo mobile) e o backend ocorre por meio de um API Gateway, que centraliza o tráfego, garante segurança e padroniza o acesso às APIs.
Os microserviços principais — UserAPI, WalletAPI, CurrencyAPI e ChatbotAPI — operam de forma desacoplada, cada um com sua própria lógica de negócio e seu próprio banco de dados (SQLite). Os serviços se comunicam por requisições REST e, para operações assíncronas, utilizam o mecanismo de mensageria RabbitMQ, que permite o envio e consumo de eventos como depósitos e comandos vindos do chatbot.
Essa arquitetura facilita escalabilidade, manutenção e isolamento de falhas, além de permitir evolução independente dos módulos.

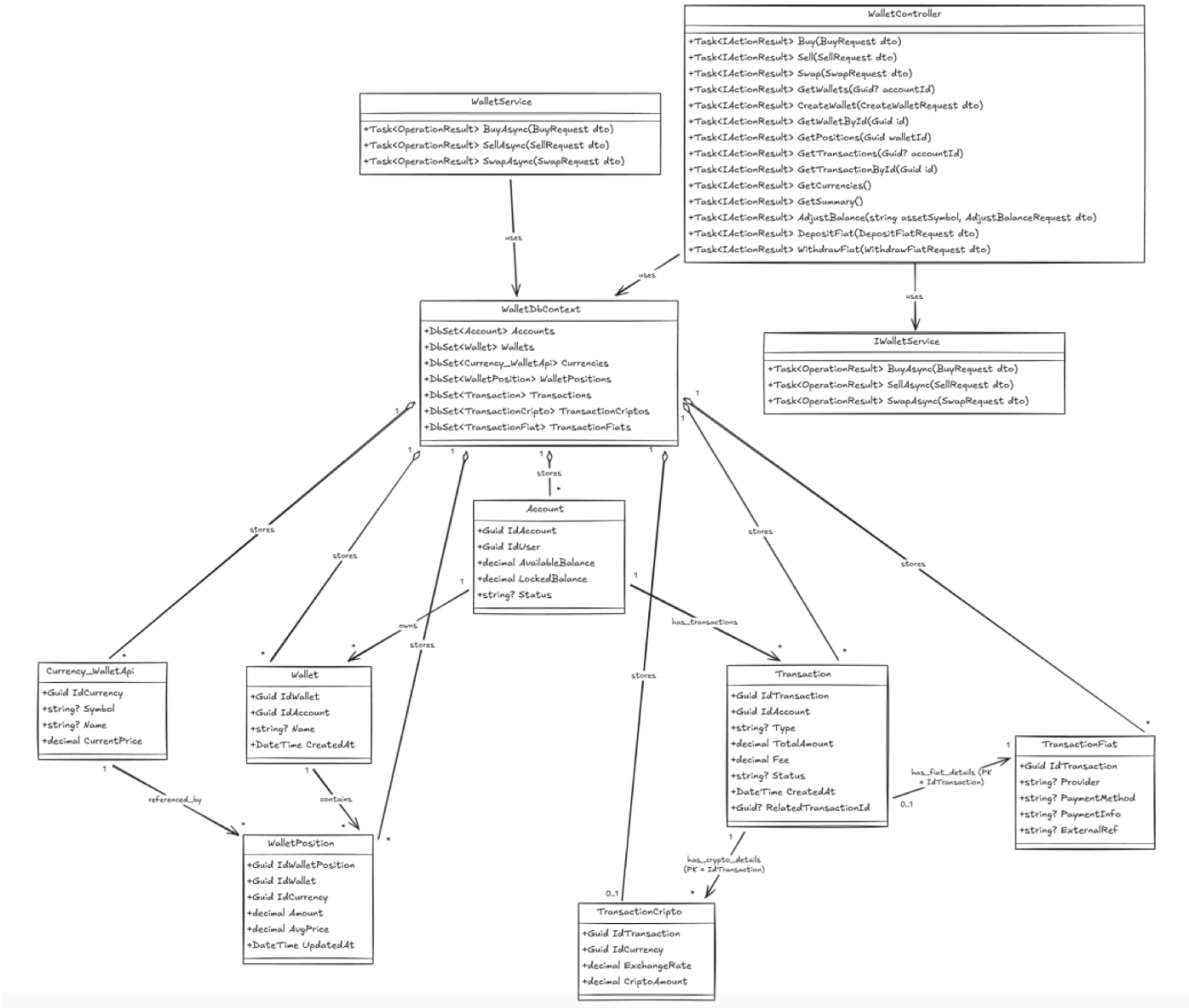


2. Diagramas de Classes (UML)

Descrição WalletAPI -

O WalletAPI gerencia as carteiras dos usuários e todas as movimentações financeiras, como depósitos, saques e transações. O WalletService centraliza as regras de negócio, incluindo atualização de saldo, registro de transações e emissão de eventos para o RabbitMQ.

WalletApi:

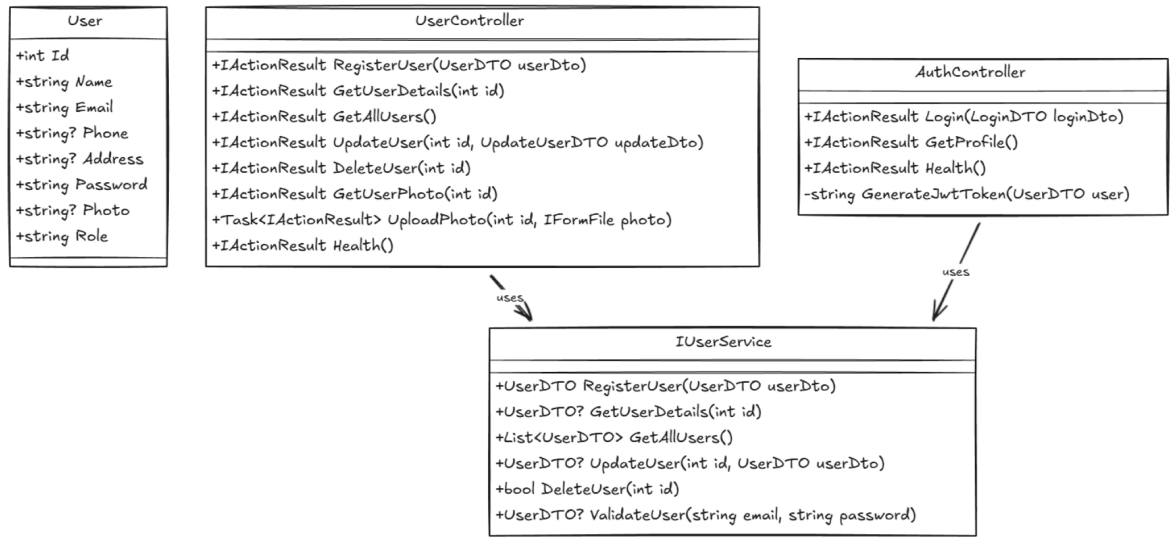


Descrição UserAPI -

O UserAPI é responsável pela gestão de usuários, incluindo cadastro, autenticação e geração de tokens JWT.

Seu diagrama de classes apresenta a entidade User, que representa os dados essenciais do usuário, e as camadas de serviço e repositório, responsáveis por encapsular regras de negócio e manipulação de dados.

UserApi:

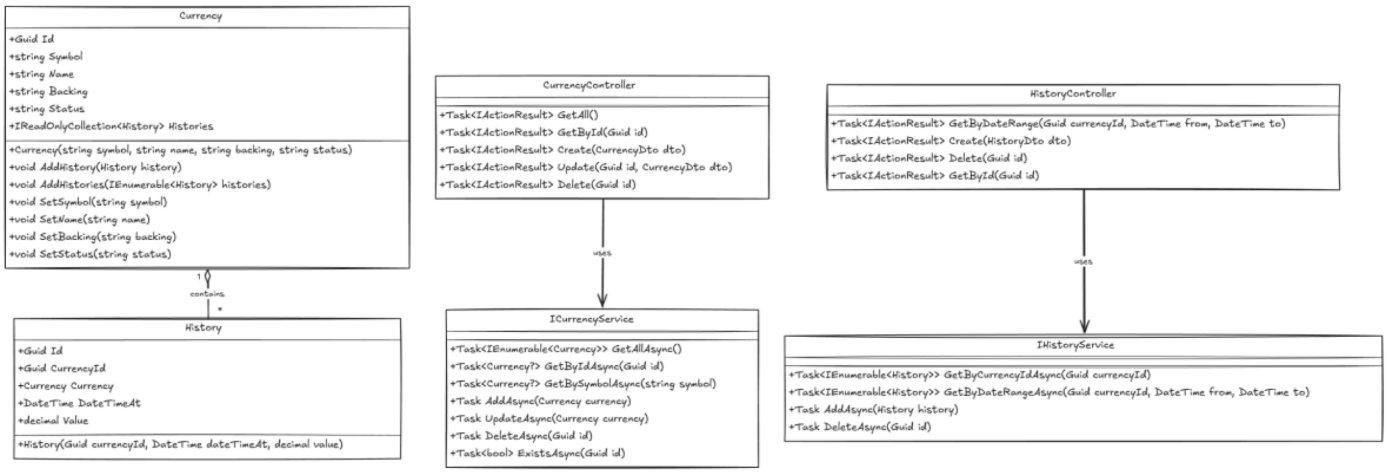


Descrição Cerrency -

O CurrencyAPI é responsável pelos dados de moedas disponíveis, mantidos na entidade CurrencyAvailables, que define informações como símbolo e preço atual.

Seu serviço central (CurrencyService) gerencia atualizações, consultas de preço e fornecimento de dados para operações de trade realizadas por outros serviços.

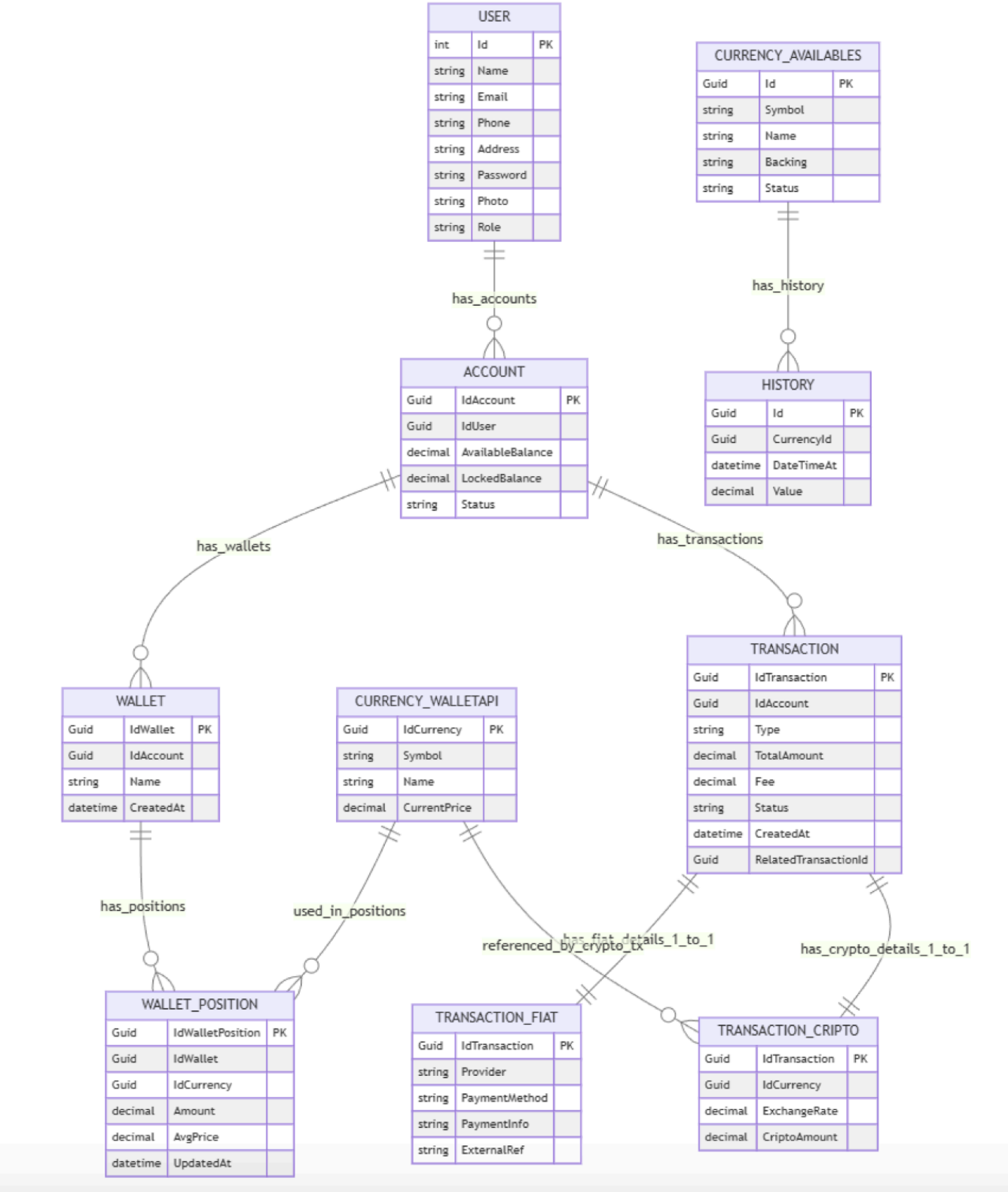
CurrencyAvailables:



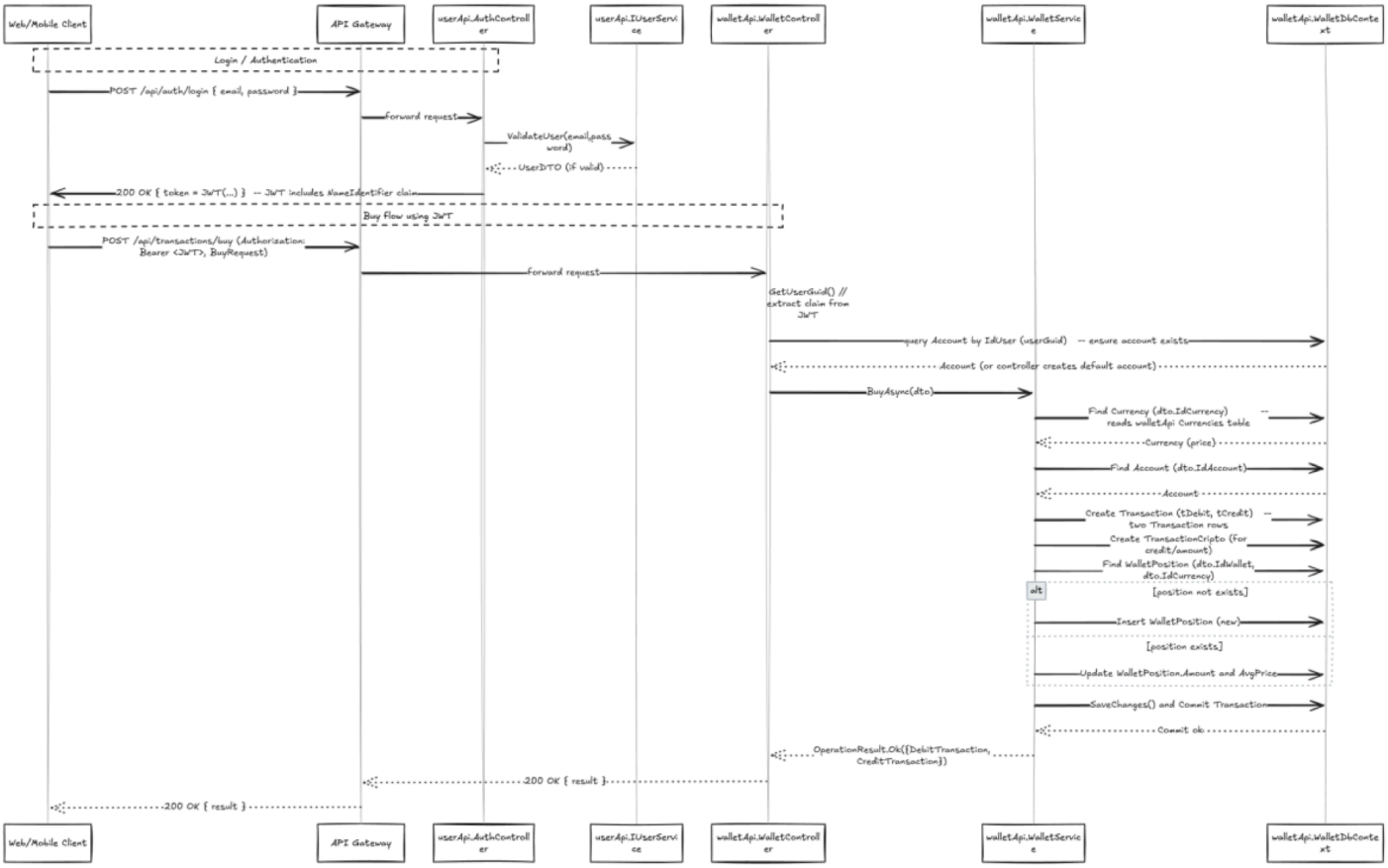
3. DER — Diagrama Entidade-Relacionamento

Descrição do Modelo de Dados -

Cada microserviço possui sua própria base de dados, seguindo o padrão de arquitetura orientada a domínios independentes.



4. Diagramas de Sequência



Descrição Login -

O fluxo de login inicia no cliente, que envia credenciais ao API Gateway.

O gateway encaminha a requisição ao UserAPI, onde os dados são validados. Após autenticação bem-sucedida, o sistema gera um token JWT e devolve ao cliente.

Esse fluxo ilustra a validação síncrona entre cliente → gateway → serviço.

Descrição Depósito -

No depósito tradicional, o cliente solicita a operação via API Gateway.

A WalletAPI registra a transação, atualiza o saldo da carteira e publica um evento no RabbitMQ informando que um depósito foi criado.

Esse evento pode acionar outros serviços interessados em monitorar atividades financeiras.

Descrição Trade -

O trade envolve interação entre WalletAPI e CurrencyAPI.

Ao solicitar a conversão de ativos, a WalletAPI consulta os preços no CurrencyAPI, realiza os cálculos necessários, atualiza o saldo e registra a transação.

O fluxo demonstra comunicação síncrona entre microserviços e aplicação de lógica financeira.

Descrição Depósito via Chatbot -

O usuário envia um comando ao chatbot; o ChatbotAPI interpreta a mensagem e publica um evento no RabbitMQ.

A WalletAPI consome esse evento, processa o depósito e atualiza a carteira.

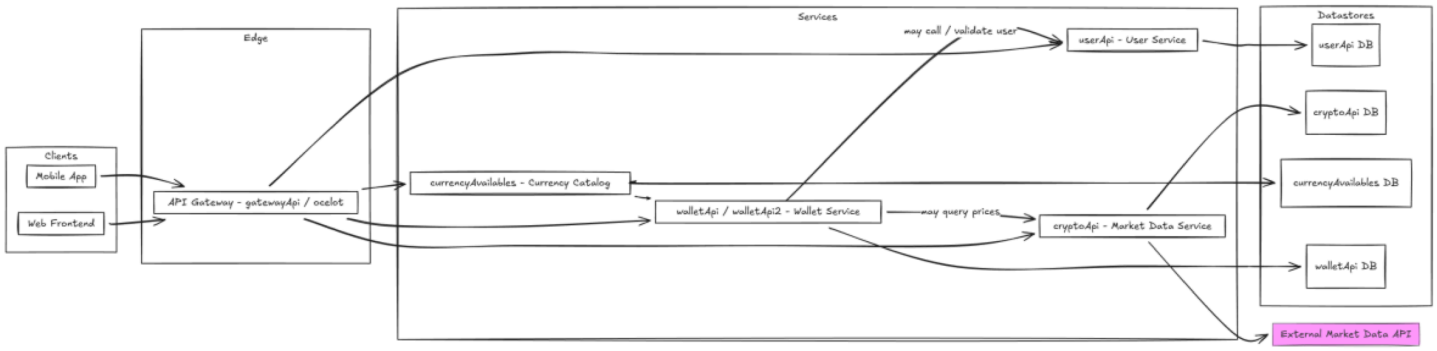
Depois, o ChatbotAPI envia retorno confirmando a operação, fechando o fluxo assíncrono.

5. Fluxo de Comunicação entre Serviços

O sistema combina comunicação síncrona e assíncrona entre microserviços.

Chamadas REST são utilizadas para operações que exigem retorno imediato, como login, consulta de saldo e execução de trades.

Já ações que podem ser processadas de forma desacoplada utilizam o RabbitMQ, por meio de exchanges e routing keys, permitindo fluxo assíncrono entre APIs.



6. Descrição textual dos componentes e tecnologias

6.1. Visão Geral da Aplicação

A aplicação proposta simula o funcionamento básico de uma corretora de criptomoedas, permitindo que usuários realizem cadastro, autenticação, consulta de saldo, depósitos, saques, trades (trocas de ativos) e interajam com um chatbot integrado.

O sistema segue o conceito de arquitetura distribuída baseada em microserviços, garantindo flexibilidade, fácil manutenção e escalabilidade.

Cada parte do sistema é independente, comunicando-se por APIs REST e, quando necessário, por mensageria assíncrona usando RabbitMQ.

6.2. Componentes da Arquitetura

A aplicação é composta por cinco microserviços principais, um API Gateway, um frontend web, um aplicativo mobilee um chatbot.

A seguir está a descrição individual de cada componente.

6.2.1 UserAPI — Serviço de Usuários

Função:

Gerencia o ciclo de vida do usuário, incluindo cadastro, login e autenticação por JWT.

Principais funcionalidades:

- Cadastro de novos usuários
- Login e geração de token JWT
- Hash seguro de senhas com BCrypt
- Controle básico de perfil

Tecnologias:

- .NET 8
- Clean Architecture
- SQLite
- BCrypt.Net

6.2.2 WalletAPI — Serviço de Carteiras e Transações

Função:

Controla saldos, depósitos, saques e operações de trade entre ativos.

Principais funcionalidades:

- Retorno de saldo total e por carteira
- Depósitos simulados
- Saques (opcional)
- Execução de trade entre criptomoedas
- Emissão de eventos no RabbitMQ (ex.: depósito concluído)

Tecnologias:

- .NET 8 com Clean Architecture
- SQLite
- RabbitMQ (publisher de eventos)

6.2.3 CurrencyAPI — Serviço de Preços e Ativos

Função:

Fornecer dados de cotações de criptomoedas e histórico para gráficos.

Principais funcionalidades:

- Retornar preço atual de um ativo (simulado ou fixo)
- Gerar histórico de preços (24h, 7d)
- Publicar eventos de atualização de preço no RabbitMQ

Tecnologias:

- .NET 8
- Banco de dados leve (SQLite ou dados em memória)
- RabbitMQ

6.2.4 ChatbotAPI — Serviço de Chat Automatizado

Função:

Permitir interação do usuário com um chatbot inteligente, simulando comandos de carteira.

Principais funcionalidades:

- Receber perguntas (“qual meu saldo?”)
- Interpretar comandos (“depositar 200 USD”)
- Publicar comandos no RabbitMQ
- Retornar respostas formatadas para o frontend

Tecnologias:

- Python (Flask)
- RabbitMQ (publisher)

- NLP simples para interpretação de texto

6.2.5 GatewayAPI — API Gateway da Plataforma

Função:

Atuar como porta de entrada única da aplicação, encaminhando chamadas externas para os microserviços corretos.

Principais funcionalidades:

- Balanceamento básico / roteamento interno
- Middleware de autenticação (JWT)
- Segurança e padronização de respostas
- Simplificar o consumo pelo frontend e mobile

Tecnologias:

- .NET 8
- JWT Authentication
- Reverse Proxy (YARP ou middleware nativo)

6.3. Frontend e Aplicativo Mobile

6.3.1 Frontend Web (Next.js + TypeScript)

Função:

Interface principal de navegação da plataforma.

Principais páginas:

- Tela de Login
- Dashboard de Saldo
- Tela de Trade
- Lista de Ativos
- Histórico de transações
- Chat com o chatbot

Tecnologias:

- Next.js (React)
- Tailwind CSS
- JWT via LocalStorage
- Axios para comunicação
- Websocket (opcional para atualizações ao vivo)

6.3.2 Aplicativo Mobile (React Native + Expo)

Função:

Versão mobile simplificada para consulta rápida de saldo e execução de ações básicas.

Principais funcionalidades:

- Login
- Saldo do usuário
- Depósitos/saques simples
- Acesso rápido ao chatbot

Tecnologias:

- React Native
- Expo
- Context API / Redux (opcional)
- Comunicação via API Gateway

6.4. Componentes de Mensageria (RabbitMQ)

O projeto utiliza RabbitMQ para troca de eventos assíncronos entre serviços.

Por que usar RabbitMQ?

- Evita acoplamento direto entre microserviços
- Permite execução de ações em background
- Facilita rastreamento e auditoria

Principais eventos emitidos:

Origem	Evento	Destino	Função
UserAPI	user.auth.success	Wallet, Chatbot	Notificar login concluído
WalletAPI	wallet.deposit.success	Chatbot	Confirmar depósito
WalletAPI	wallet.trade.success	Currency, Chatbot	Confirmar trade
CurrencyAPI	currency.price.update	Wallet, Chatbot	Atualização de preço
ChatbotAPI	chatbot.wallet.deposit	Wallet	Solicitação de depósito via chat

6.5. Segurança e Autenticação

O sistema usa JWT para autenticação em todas as rotas protegidas.

Medidas de segurança implementadas:

- Hash de senha com BCrypt
- Token assinado com chave secreta
- Validação de token no GatewayAPI
- Sanitização de entradas
- Logs de ações sensíveis (login, trade, depósito)

6.6. Clean Architecture dentro dos Microserviços

Todos os microserviços seguem a estrutura:

```
/API
/Domain
/Infrastructure
/Application
```

Objetivo:

Separar responsabilidades e facilitar manutenção.

Resumo das camadas:

- **API:** rotas e controllers
- **Domain:** entidades do negócio
- **Infrastructure:** banco e repositórios
- **Application:** casos de uso e regras de negócio

6.7. Banco de Dados — SQLite

Cada microserviço possui seu próprio banco SQLite, garantindo isolamento e independência.

Vantagens:

- Fácil de rodar localmente
- Sem dependência externa
- Ideal para MVP

6.8. Controle de Versão — GitHub + GitFlow

O projeto utiliza GitHub como controle de versão, com o fluxo:

- **main** — versão estável
- **develop** — versão em desenvolvimento
- **feature/** — novas funcionalidades
- **release/** — preparação para entrega

- **hotfix/** — correções rápidas

6.9. Tecnologias Principais e Justificativa de Uso

Tecnologia	Função	Justificativa
.NET 8	Backend	Performance, robustez e facilidade de criar APIs
Next.js	Frontend	SSR/SPA, escalável e moderno
Tailwind CSS	Estilos	Produtividade e consistência visual
React Native	Mobile	Código compartilhado e rápida entrega
RabbitMQ	Mensageria	Comunicação assíncrona confiável
Flask / Python	Chatbot	Agilidade em NLP e integrações
SQLite	Bancos	Simples, portátil, ideal para MVP
GitHub	Versionamento	Colaboração e rastreamento