Outubro de 2025

Sobre o arc42

arc42, o template para documentação de software e arquitetura de sistemas.

Versão do template 8.2 PT. (baseado na versão AsciiDoc), Setembro de 2024

Criado, mantido e © pelo Dr. Peter Hruschka, Dr. Gernot Starke e colaboradores. Veja https://arc42.org.

1. Introdução e Objetivos

O Whale é um sistema desenvolvido para gerenciamento de investimentos em criptomoedas. Seu objetivo principal é permitir que investidores registrem seus ativos, acompanhem a valorização ou desvalorização de acordo com o valor investido e visualizem o desempenho utilizando dados de cotação com atualizações periódicas. Além de oferecer um controle prático do portfólio, o Whale busca entregar uma experiência clara, acessível e visualmente atrativa, tanto para investidores iniciantes quanto para usuários mais experientes.

1.1 Visão Geral dos Requisitos

Os principais requisitos funcionais do Whale são:

- Cadastro de investidor: Registro de usuários com dados pessoais (nome e e-mail).
- Uso anônimo com importação/exportação JSON: Possibilidade de usar o sistema sem login, enviando dados em JSON e exportando-os no mesmo formato.
- Registro de criptomoedas: Inclusão de informações de compra (moeda, quantidade, data e valor investido).
- Registro de saques/vendas: Controle de retiradas ou vendas parciais/totais, atualizando automaticamente o saldo.
- Integração com API de cotações em com atualizações periódicas: Atualização automática dos preços de mercado a cada 30 segundos.

1.2 Objetivos de Qualidade

Número	Qualidade	Motivação
1	Confiabilidade	O sistema deve calcular corretamente ganhos e perdas, sem inconsistências.
2	Usabilidade	Interface clara e intuitiva para que investidores de qualquer nível possam utilizá-la.
3	Interoperabilidade	Suporte a importação/exportação JSON e API para integração com outros sistemas.
4	Escalabilidade	Suportar aumento de usuários e ativos sem degradação de desempenho.

Table 1 - Objetivos de Qualidade

1.3 Partes Interessadas

Função/Nome	Contato	Expectativas
Usuários	E-mail	Gerenciar seu portfólio de criptomoedas e acompanhar desempenho.
Desenvolvedores	E-mail, Microsoft Teams	Evoluir o sistema, garantindo qualidade do código e integração com APIs externas.
Arquitetos de Software	E-mail, Microsoft Teams	Avaliar e propor melhorias na arquitetura e escalabilidade.

Table 2 - Partes Interessadas

2. Restrições Arquiteturais

Código	Restrição	Motivação
CT1	Implementação em .NET para microserviços e funções e Node.js para o BFF.	Garantir robustez, modularidade e integração com APIs e funções.
CT2	Frontend React com microfrontend.	Fornecer experiência fluida, responsiva e multiplataforma.
CT3	Persistência em banco relacional (SQL Server) e não relacional (Atlas Mongo DB)	Garantir consistência e escalabilidade dos dados.
CT4	Integração com APIs externas de cotação com atualizações periódicas, de 30 em 30 segundos (como o Binance API).	Atualização de preços precisa e automática.

Código	Restrição	Motivação
CT5	Exportação/importação em JSON.	Interoperabilidade e facilidade de backup/uso anônimo.

Table 3 - Restrições Técnicas

Código	Restrição	Motivação
CN1	Equipe pequena de desenvolvimento.	Foco em simplicidade e modularidade da arquitetura.
CN2	Tempo limitado de entrega.	Funcionalidades principais devem ter prioridade.
CN3	Repositório Git com versionamento e CI/CD.	Garantir qualidade e rastreabilidade.

Table 4 - Restrições de Negócio

3. Contexto e Escopo

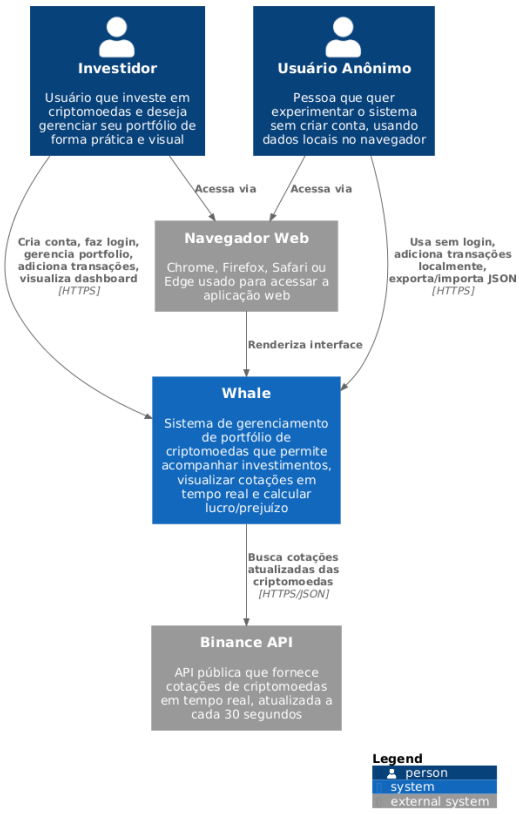

Diagram 1 - C4 Model Nível 1 (contexto)

3.1 Contexto Negocial

O Whale é usado por investidores que desejam controlar seus ativos de criptomoedas. Ele depende de:

- APIs de mercado (Binance) para obter cotações com atualizações periódicas de 30 em 30 segundos.
- Navegador web para acesso a aplicação

Fluxo principal:

1. O investidor cadastra ativos ou importa dados em JSON.
2. O sistema consulta cotações de 30 em 30 segundos.
3. O sistema exibe um resumo e o desempenho do portfolio

3.2 Contexto Técnico

- Backend: Implementado em .Net, responsável por persistência, regras de negócio e integração com APIs externas. Dividido em dois microserviços, um BFF e duas funções.
- Frontend: Interface web responsiva em React Native com microfrontend, comunicando-se com o backend via BFF.
- Banco de Dados: Relacional, armazena o dado de usuário. Não relacional, armazena dados do portfolio e transações.
- Serviços externos: APIs de cotação.

4. Estratégia de Solução

A estratégia do Whale combina simplicidade arquitetural com robustez tecnológica, adotando uma arquitetura de microserviços com Backend for Frontend (BFF) e microfrontends. O domínio central envolve Investidor, Criptomoeda, Portfolio e Transações com cotações atualizadas periodicamente a cada 30 segundos. A estratégia do Whale combina uma arquitetura moderna de microserviços com processamento assíncrono, onde o frontend consiste em uma Shell Application React com Micro Frontend (Dashboard MFE) carregado via iframe para exibir as Top 12 criptomoedas, enquanto o backend utiliza um BFF (Backend for Frontend) em Node.js + Express responsável por agregação de dados, proxy de requisições, orquestração de eventos, validação de import/export JSON e autenticação JWT, dois microserviços em .NET - Users Service (Azure SQL Server) para gerenciamento de usuários e autenticação, e Portfolio Service (MongoDB Atlas) para CRUD de portfolios e transações com cálculos de performance - e Azure Functions serverless para buscar as Top 12 criptomoedas da Binance (HTTP Trigger, atualização a cada 30s) e processar transações de forma assíncrona via eventos (Event Trigger), garantindo escalabilidade, separação de responsabilidades e melhor performance através do desacoplamento entre frontend, orquestração e processamento de dados.

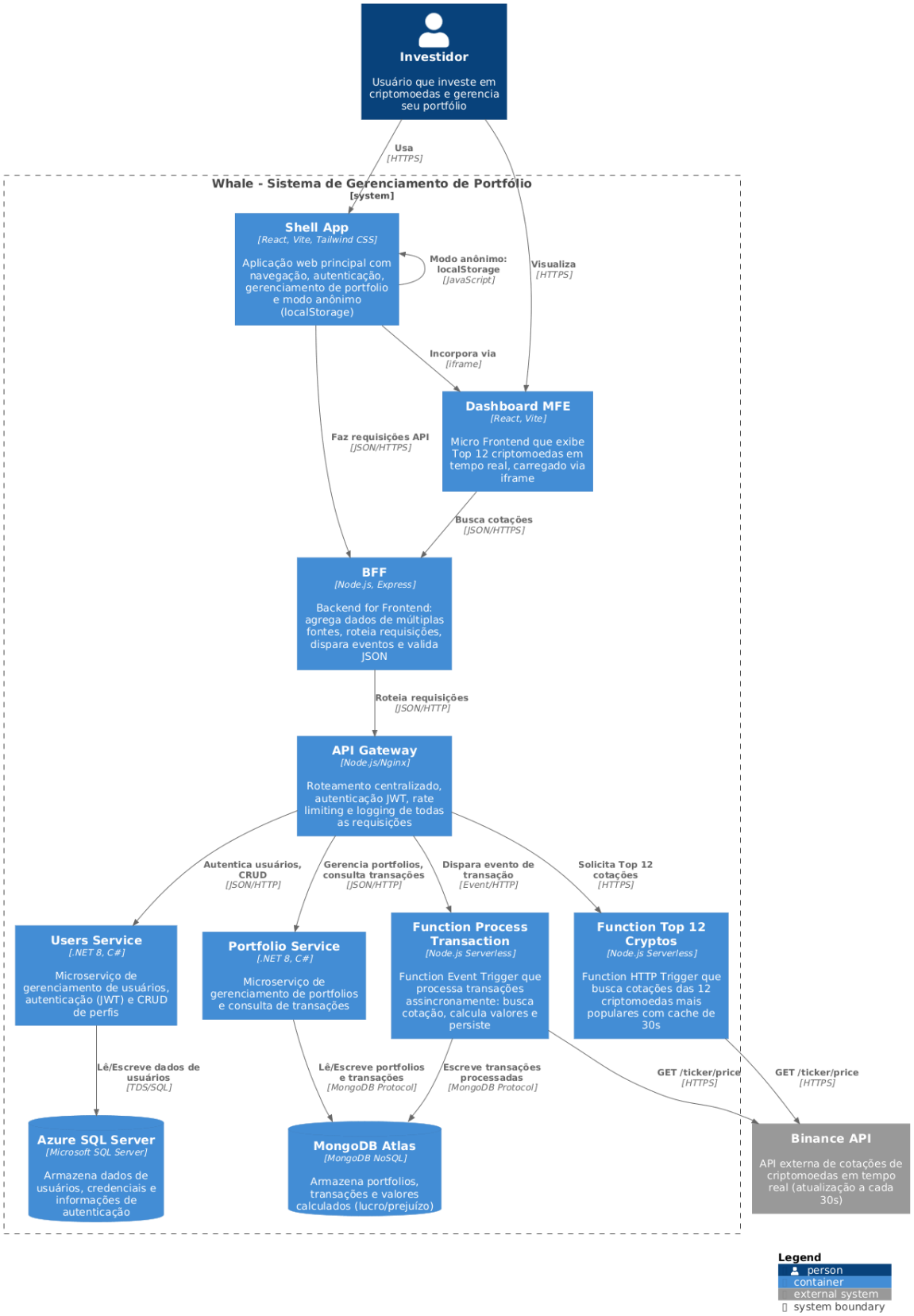

Diagram 2 - C4 Model Nível 2 (container)

5. Visão de Blocos de Construção

5.1 Visão Sistêmica Geral de Caixa Branca

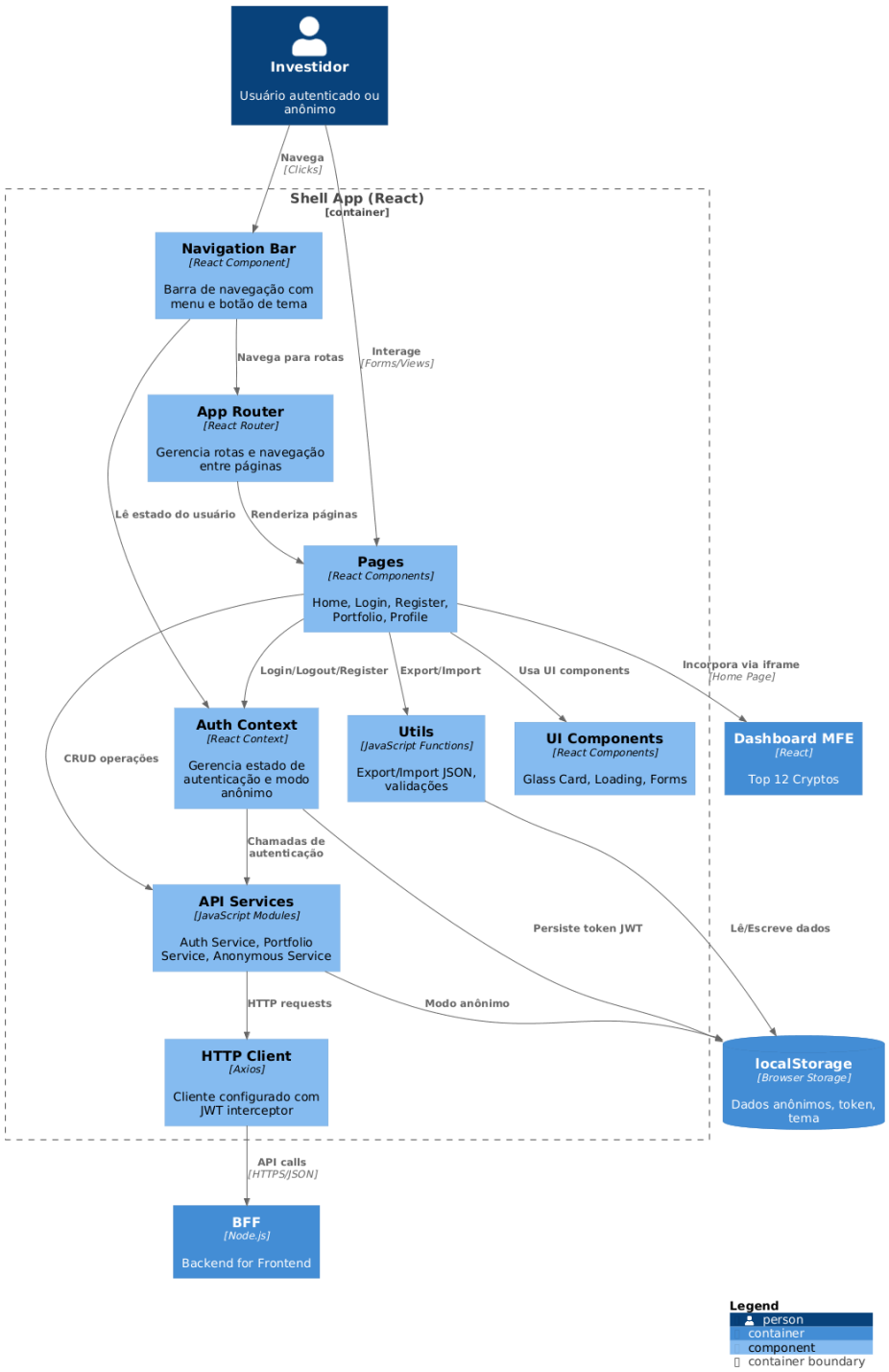

Diagrama 3 - C4 Model nível 3 (component) – Shell App

Motivação

O sistema Whale foi decomposto em blocos principais para facilitar manutenção e escalabilidade. O frontend é uma Shell Application React com Micro Frontend (Dashboard MFE) carregado via iframe, que interage com o backend através de um BFF (Backend for Frontend) desenvolvido em Node.js + Express, responsável por agregar dados, orquestrar eventos e gerenciar autenticação JWT. Dois microserviços em .NET implementam responsabilidades isoladas: Users Service (autenticação e gestão de usuários) e Portfolio Service (gestão de portfolios e transações). O armazenamento utiliza polyglot persistence com Azure SQL Server para usuários e MongoDB Atlas para portfolios e transações. O sistema integra-se com Azure Functions serverless para buscar Top 12 criptomoedas da Binance (HTTP Trigger com atualização a cada 30 segundos) e processar transações de forma assíncrona (Event Trigger), além de suportar modo anônimo com import/export JSON validado pelo BFF.

Blocos de Construção Contidos

- Shell Application React: Aplicação principal responsável pela interface web ao usuário, implementando glassmorphism com gradiente azul animado e suporte a modo claro/escuro. É responsiva e multiplataforma, comunicando-se com o BFF via HTTP/JSON.API Gateway: Roteia e autêntica as requisições. Consome os serviços backend e os disponibiliza para a interface Web App.
- Dashboard MFE (Micro Frontend): Micro Frontend carregado via iframe que exibe as Top 12 criptomoedas mais populares em tempo real, consumindo diretamente a Azure Function de cotações.
- BFF - Backend for Frontend (Node.js + Express): Camada intermediária que centraliza agregação de dados de múltiplos microserviços, proxy de requisições, orquestração de eventos para Azure Functions, validação de JSON para modo anônimo e gerenciamento de autenticação JWT.
- Users Service (.NET + Azure SQL Server): Microserviço responsável por gerenciar cadastro, login, validação de tokens JWT, CRUD de perfis de usuário e hash de senhas com bcrypt. Utiliza Azure SQL Server Free (1 DTU) para persistência.
- Portfolio Service (.NET + MongoDB Atlas): Microserviço responsável por CRUD de portfolios, CRUD de transações (compra/venda), cálculos de performance (lucro/prejuízo) e validação via Mongoose schemas. Utiliza MongoDB Atlas Free para persistência.
- Top 12 Cryptos Function (Azure Function - HTTP Trigger): Function serverless que busca as 12 criptomoedas mais populares da Binance, retorna array com símbolos e preços, e atualiza a cada 30 segundos. Consumida diretamente pelo Dashboard MFE.
- Process Transaction Function (Azure Function - Event Trigger): Function serverless a ser desenvolvida que recebe eventos de criação de transação via BFF, busca cotação atual da moeda, calcula valores (investido vs atual, lucro/prejuízo) e persiste no MongoDB via Portfolio Service de forma assíncrona.
- Azure SQL Server (Banco de Usuários).

- MongoDB Atlas (Banco de Portfolios).

Interfaces Importantes

- BFF ⇔ Function
- Function ⇔ Binance API (REST externo)
- API Gateway ⇔ Microserviços de usuário e portfolio
- Microserviço de usuário ⇔ Azure SQL Server
- Microserviço de portfolio ⇔ Atlas MongoDB

Caixa Preta 1: Shell Application React

Interface web responsiva com o usuário final. Permite login, cadastro, visualização de portfólio, adição/remoção de ativos, importação/exportação JSON e navegação entre módulos. Implementa glassmorphism com gradiente azul animado e suporte a modo claro/escuro.

Interface(s): API REST exposta pelo BFF via HTTP/JSON.

Requisitos Cumpridos: Usabilidade, interoperabilidade.

Caixa Preta 2: API Gateway

Roteamento e autenticação centralizada das chamadas da aplicação.

Interface(s):

- REST API pública consumida pelo Web App.
- REST APIs internas para comunicação com os serviços backend.

Características de Qualidade/Desempenho: Segurança, padronização de acesso.

Requisitos Cumpridos: Confiabilidade, escalabilidade.

Problemas/Riscos Abertos: Ponto único de falha caso não seja replicado.

Caixa Preta 3: Dashboard MFE (Micro Frontend)

Micro Frontend carregado via iframe que exibe as Top 12 criptomoedas mais populares em tempo real. Consome diretamente a Azure Function de cotações a cada 30 segundos.

Interface(s): HTTP Trigger da Azure Function Top 12 Cryptos.

Requisitos Cumpridos: Performance, usabilidade, dados atualizados em tempo real.

Caixa Preta 4: Backend for Frontend (Node.js + Express)

Camada intermediária que centraliza agregação de dados de múltiplos microserviços, proxy de requisições, orquestração de eventos para Azure Functions, validação de JSON para modo anônimo e gerenciamento de autenticação JWT.

Interfaces: REST API pública consumida pelo Shell Application, REST APIs internas para comunicação com Users Service e Portfolio Service (.NET) e HTTP e Event-based triggers para Azure Functions.

Características de Qualidade/Desempenho: Segurança (JWT), padronização de acesso, agregação de dados.

Requisitos Cumpridos: Confiabilidade, escalabilidade, interoperabilidade.

Problemas/Riscos Abertos: Ponto único de falha caso não seja replicado.

Caixa Preta 5: Users Service (.NET + Azure SQL Server)

Microserviço em .NET responsável por gerenciar cadastro, login, validação de tokens JWT, CRUD de perfis de usuário e hash de senhas com bcrypt.

Interface(s): Endpoints: /register, /login, /users (CRUD). Integração com Azure SQL Server Free (1 DTU)

Requisitos Cumpridos: Confiabilidade, segurana, autenticação.

Caixa Preta 6: Portfolio Service (.NET + MongoDB Atlas)

Microserviço em .NET responsável por CRUD de portfolios, CRUD de transações (compra/venda), cálculos de performance (lucro/prejuízo) e validação via Mongoose schemas. Integra-se com Binance API para cotações.

Interfaces:

- Endpoints: /portfolios, /transactions
- Integração com Binance API (REST externo)
- Integração com MongoDB Atlas Free

Problemas/Riscos Abertos: Limite de requisições na API externa Binance.

Caixa Preta 7: Top 12 Cryptos Function (Azure Function - HTTP Trigger)

Function serverless que busca as 12 criptomoedas mais populares da Binance, retorna array com símbolos e preços, e atualiza a cada 30 segundos. Consumida diretamente pelo Dashboard MFE.

Interfaces:

- HTTP Trigger público (GET)
- Integração com Binance API

Requisitos Cumpridos: Performance, dados atualizados periodicamente, baixa latência.

Caixa Preta 7: Process Transaction Function (Azure Function - Event Trigger)

Function serverless a ser desenvolvida que recebe eventos de criação de transação via BFF, busca cotação atual da moeda na Binance, calcula valores (investido vs atual, lucro/prejuízo) e persiste no MongoDB via Portfolio Service de forma assíncrona.

Interfaces:

- Event Trigger disparado pelo BFF
- Integração com Binance API
- Integração com Portfolio Service (.NET)

Requisitos Cumpridos: Escalabilidade, processamento assíncrono, melhor performance.

Caixa Preta 8: Azure SQL Server (Banco de Usuários)

Persiste dados de usuários, credenciais com hash bcrypt e perfis. Utiliza plano Free (1 DTU).

Interfaces: Conexão direta via Entity Framework Core no Users Service

Caixa Preta 9: MongoDB Atlas (Banco de Portfolios)

Persiste portfolios, transações e cálculos de performance com schemas Mongoose. Utiliza plano Free.

Interfaces: Conexão direta via driver MongoDB no Portfolio Service.

5.2 Nível 2 – Exemplos de Caixas Brancas Detalhadas

Caixa Branca – BFF (Backend For Frontends) (Detalhe)

Componentes: Controlador de Rotas, Serviço de Agregação, Cliente HTTP, Despachante de Eventos, Validador JSON.

Interfaces:

- Endpoints REST públicos consumidos pelo frontend
- Integração HTTP com Users Service (Azure SQL)
- Integração HTTP com Portfolio Service (MongoDB)
- Integração HTTP com Functions (Top 12 e Process Transaction)
- Middleware de autenticação JWT.

Sub-blocos:

- Roteador: Gerencia todas as rotas da aplicação (/auth, /users, /portfolio, /transactions, /aggregation, /anonymous)
- Agregador de dados: Realiza agregação de dados buscando informações de múltiplas fontes (microserviços + functions) e consolidando em uma única resposta
- HTTP Client: Encapsula requisições para microserviços e functions usando axios/fetch
- Despachador de Eventos: Envia eventos para Function Process Transaction quando novas transações são criadas
- JWT Middleware: Valida tokens JWT em rotas protegidas

- Gerenciador JSON: Gerencia validação de estrutura JSON para import/export de portfolios anônimos

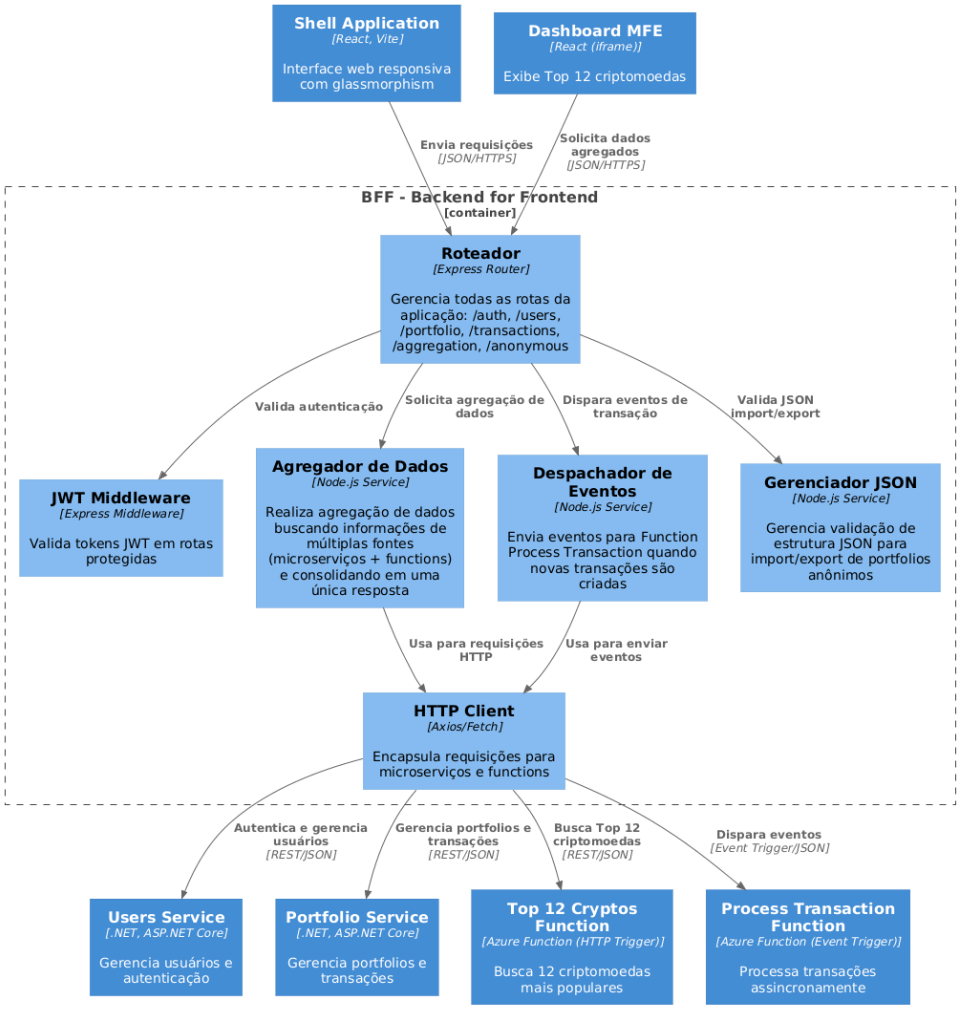

Diagram 4 - C4 Model nível 3 (component) – BFF

Caixa Branca – Serviço de Usuários (Detalhe)

Componentes: Controlador REST, Gerenciador de Autenticação, Password Hasher, Repositório de Usuários.

Interfaces:

- Endpoints de autenticação: /auth/login, /auth/register
- Endpoints CRUD de usuários: /users (GET, POST, PUT, DELETE)
- Integração com Azure SQL Server via Entity Framework Core

Sub-blocos:

- Auth Controller: Recebe requisições de login e registro, valida credenciais e retorna JWT

- Users Controller: Gerencia operações CRUD de perfis de usuário
- Authentication Service: Lógica de negócio para autenticação (validação de credenciais, geração de tokens)
- Users Service: Lógica de negócio para operações de usuários (validação de dados, regras de negócio)
- Password Hasher: Utiliza BCrypt.NET para hash e verificação de senhas com salt configurável
- JWT Manager: Gera e valida JSON Web Tokens com System.IdentityModel.Tokens e tempo de expiração configurável
- User Repository: Encapsula acesso ao Azure SQL Server usando Entity Framework Core com padrão Repository

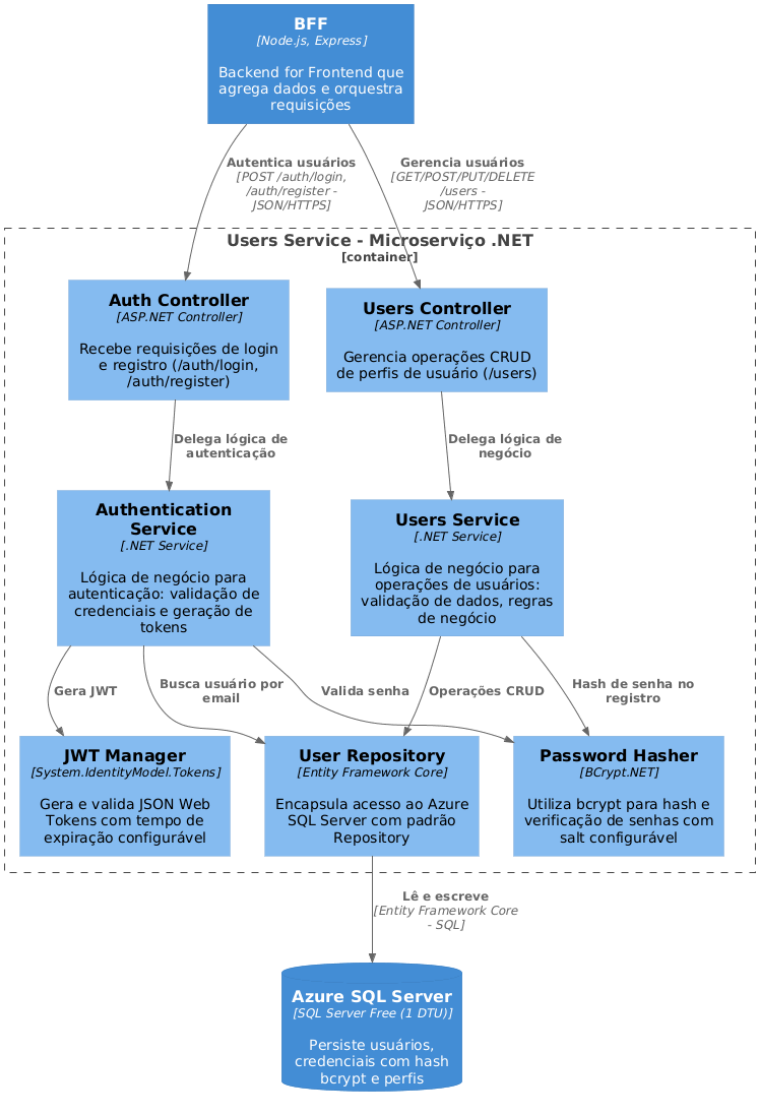

Diagram 5 - C4 Model nível 3 (component) - Gerenciamento de Ativos

Caixa Branca – Serviço de Portfolio (Portfolio Service)

Componentes: Controlador REST, Módulo de Negócio Portfolio, Módulo de Transações, Repositório MongoDB.

Interfaces:

- Endpoints de portfolio: /portfolios (GET, POST)
- Endpoints de transações: /transactions (GET, POST, DELETE)
- Integração com MongoDB Atlas via MongoDB.Driver (.NET)
- Integração com Binance API para cotações em tempo real

Sub-blocos:

- Portfolio Controller: Recebe requisições REST para operações de portfolio (criar, listar, buscar)
- Transaction Controller: Gerencia operações de transações (criar, listar, deletar)
- Portfolio Service: Lógica de negócio para cálculos de portfolio (valor total investido, valor atual, performance/lucro/prejuízo)
- Transaction Service: Aplica regras de negócio para compra/venda de criptomoedas e validações
- Price Fetcher: Busca cotações de criptomoedas da Binance API em tempo real usando HttpClient
- Portfolio Repository: Encapsula acesso ao MongoDB usando MongoDB.Driver (.NET) com padrão Repository
- Transaction Repository: Persiste e consulta transações no MongoDB usando MongoDB.Driver

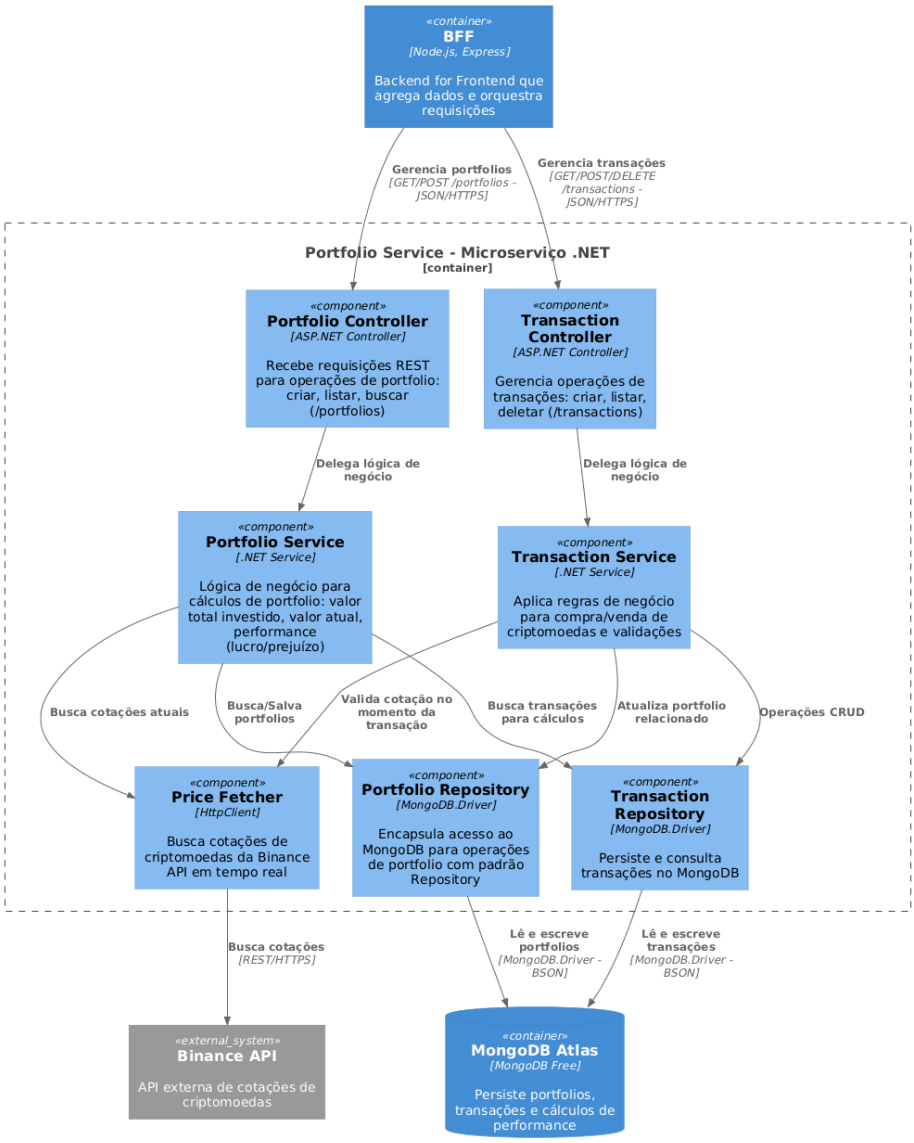

Diagram 6 - C4 Model nível 3 (component) – Portfolio Service

Caixa Branca – Function Process Transaction (Detalhe)

Componentes: Event Handler, Price Fetcher, Calculator, MongoDB Client.

Interfaces:

- Event Trigger disparado pelo BFF quando uma nova transação é criada
- Integração HTTP com Binance API para buscar cotações
- Integração HTTP com Portfolio Service para persistir dados calculados

Sub-blocos:

- Event Trigger Handler: Recebe eventos de criação de transação disparados pelo BFF

- Transaction Processor: Orquestra o processamento da transação (validação, busca de cotação e cálculo)
- Price Fetcher: Busca cotação atual da criptomoeda na Binance API usando HttpClient/Axios
- Calculator: Calcula valores (investido vs atual, lucro/prejuízo, percentual de retorno)
- Portfolio Service Client: Cliente HTTP para comunicação com Portfolio Service, enviando os dados calculados para persistência

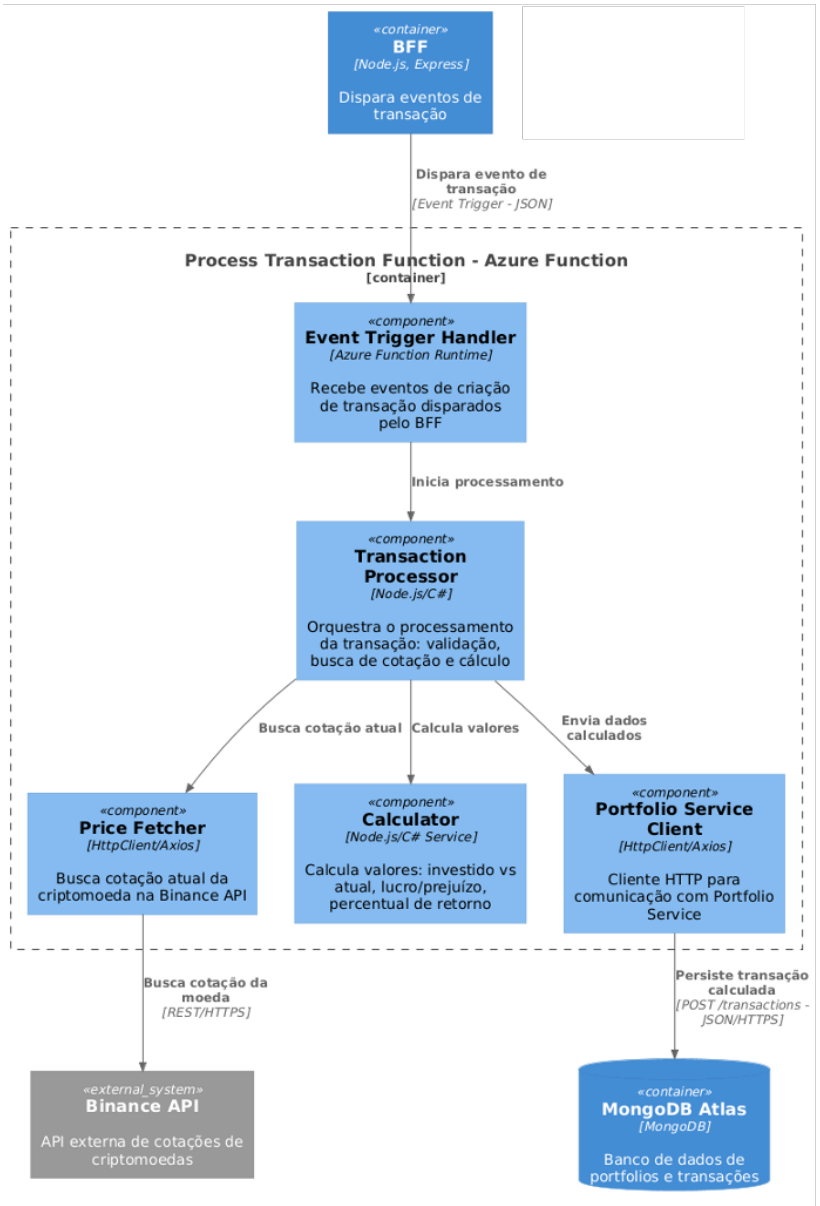

Diagram 7 - C4 Model nível 3(Component) - Transaction Function

6. Visão de Tempo de Execução

O usuário deseja adicionar uma transação de compra de criptomoeda (ex.: 0.5 BTC) ao seu portfólio. O sistema busca a cotação atual antes de criar a transação, exibe o valor para confirmação do usuário, e processa a transação de forma assíncrona, calculando lucro/prejuízo e salvando no banco de dados.

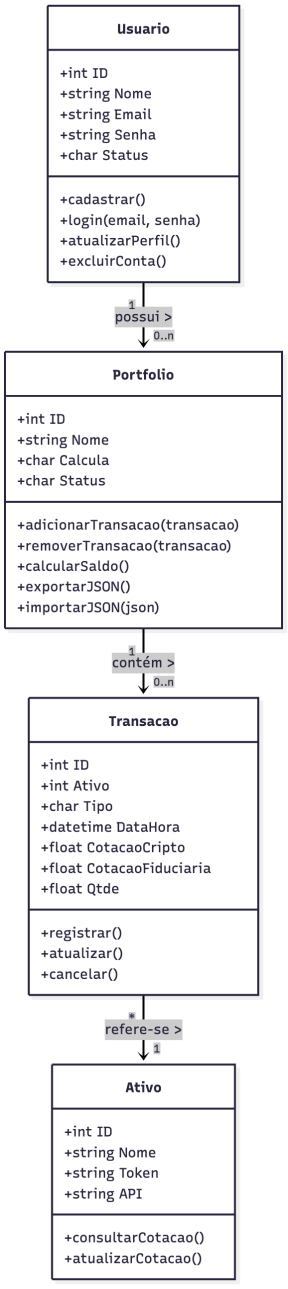

Diagram 8 - Diagrama de Classes

Cenário 1: Transação de compra

O usuário deseja adicionar uma transação de compra de criptomoeda (ex.: 0.5 BTC) ao seu portfólio. O sistema busca a cotação atual antes de criar a transação, exibe o valor para confirmação do usuário, e processa a transação de forma assíncrona, calculando lucro/prejuízo e salvando no banco de dados.

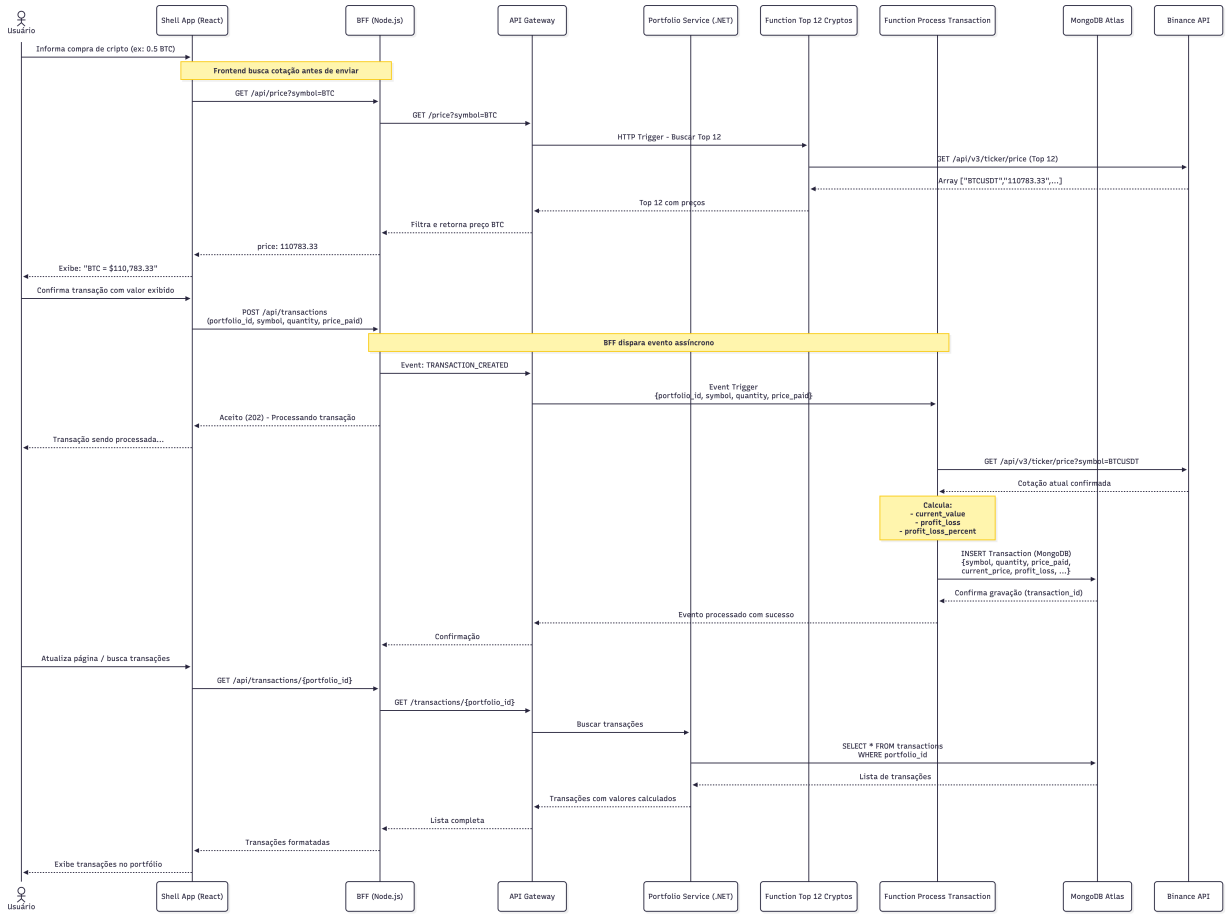

Diagram 9 - Diagrama de Sequência - Criação de portfólio e transação de compra

Fluxo:

Fase 1: Busca de Cotação Atual

1. O usuário, via Shell App (React), inicia o processo de adicionar uma transação de compra informando a criptomoeda desejada (ex.: BTC) e a quantidade.
2. Antes de criar a transação, o Shell App solicita a cotação atual ao BFF através da requisição GET /api/price?symbol=BTC.
3. O BFF encaminha a requisição ao API Gateway, que roteia para a Function Top 12 Cryptos.

4. A Function Top 12 Cryptos realiza uma consulta à API externa da Binance para obter as cotações das 12 criptomoedas mais populares.
5. A API Binance retorna um array com os dados no formato ["BTCUSDT","110783.33","ETHUSDT","3965.25",...].
6. A Function retorna os dados ao API Gateway, que filtra e extrai o preço do BTC solicitado.
7. O API Gateway retorna o preço ao BFF, que repassa para o Shell App.
8. O Shell App exibe o valor atual da criptomoeda para o usuário: "BTC = $110,783.33".

Fase 2: Criação de Transação (Processamento Assíncrono)

9. O usuário confirma a transação com o valor exibido.
10. O Shell App envia os dados da transação ao BFF através de POST /api/transactions incluindo: portfolio_id, symbol (BTC), quantity (0.5) e price_paid (valor que o usuário pagou na compra).
11. O BFF recebe a requisição e imediatamente dispara um evento assíncrono TRANSACTION_CREATED através do API Gateway.
12. O API Gateway encaminha o evento para a Function Process Transaction (Event Trigger).
13. O BFF retorna imediatamente ao Shell App com status 202 Accepted ("Transação sendo processada"), sem esperar o processamento completo.
14. O Shell App exibe ao usuário: "Transação sendo processada...".

Fase 3: Processamento pela Function (Assíncrono)

15. A Function Process Transaction recebe o evento e inicia o processamento.
16. A Function consulta novamente a API Binance para obter a cotação mais atualizada do BTC no momento do processamento.
17. A API Binance retorna a cotação atual confirmada.
18. A Function realiza os seguintes cálculos:
    - current_value = quantity × current_price
    - profit_loss = current_value - (quantity × price_paid)
    - profit_loss_percent = (profit_loss / invested_value) × 100
19. A Function conecta diretamente ao MongoDB Atlas e insere a transação com todos os dados calculados: symbol, quantity, price_paid, current_price, current_value, profit_loss, profit_loss_percent, transaction_date, processed_at.
20. O MongoDB confirma a gravação retornando o transaction_id.
21. A Function notifica o API Gateway que o evento foi processado com sucesso.
22. O API Gateway confirma ao BFF o processamento.

Fase 4: Consulta de Transações

23. O usuário atualiza a página ou navega para visualizar as transações do portfólio.
24. O Shell App solicita ao BFF a lista de transações através de GET /api/transactions/{portfolio_id}.
25. O BFF encaminha a requisição ao API Gateway.
26. O API Gateway roteia para o Portfolio Service (.NET).
27. O Portfolio Service consulta o MongoDB Atlas executando uma query para buscar todas as transações do portfólio especificado.
28. O MongoDB retorna a lista de transações com todos os valores calculados (incluindo lucro/prejuízo atual).
29. O Portfolio Service retorna as transações ao API Gateway.
30. O API Gateway repassa os dados ao BFF.
31. O BFF formata e envia a lista completa ao Shell App.
32. O Shell App renderiza a tabela de transações, exibindo para o usuário: moeda, quantidade, preço de compra, preço atual, lucro/prejuízo e percentual de valorização.

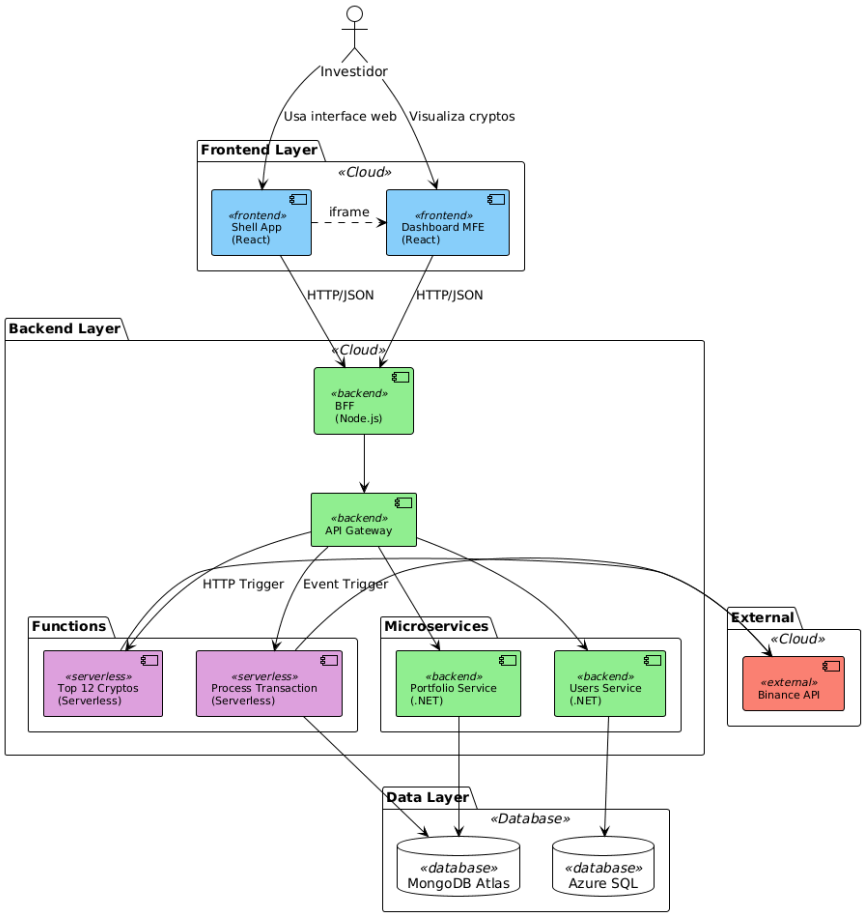

Diagram 10 - Diagrama de Componentes

7. Visão de Implantação

A visão de implantação mostra a estrutura técnica usada para executar o Whale. Ela garante que os requisitos de qualidade, como escalabilidade, confiabilidade e portabilidade, sejam atendidos. A arquitetura é baseada em containers Docker, bancos de dados em nuvem (Azure SQL e MongoDB Atlas), serverless functions e serviços externos acessados pela internet.

7.1 Nível de Infraestrutura 1 – Visão Geral

Elementos principais:
Frontend:

1. Shell App (React): Aplicação principal hospedada no Azure Web Static App.
2. Dashboard MFE (React): Micro Frontend hospedado separadamente, carregado via iframe.

Backend:

3. BFF (Node.js): Container responsável por agregação de dados, proxy e disparo de eventos.
4. API Gateway: Container responsável pelo roteamento centralizado e autenticação JWT.
5. Users Service (.NET): Container para gerenciamento de usuários e autenticação.
6. Portfolio Service (.NET): Container para gerenciamento de portfolios e consulta de transações.

Serverless Functions:

7. Function Top 12 Cryptos: Serverless function (Azure Functions/AWS Lambda/Google Cloud Functions) com HTTP Trigger.
8. Function Process Transaction: Serverless function com Event Trigger para processamento assíncrono.

Banco de Dados:

9. Azure SQL Server (Free 1 DTU): Banco relacional para dados de usuários e credenciais.
10. MongoDB Atlas (Free Tier): Banco NoSQL para portfolios e transações.

APIs Externas:

11. Binance API: Fonte de cotações de criptomoedas com atualização a cada 30 segundos.

Características de Qualidade e/ou Desempenho

12. Escalabilidade Horizontal: O BFF, API Gateway e microserviços são stateless e podem ser replicados em múltiplos containers. Functions escalam automaticamente conforme demanda.
13. Portabilidade: Todo o backend é containerizado com Docker, reduzindo dependências de ambiente. Frontend pode ser deployado em qualquer servidor web.

14. Confiabilidade: Bancos de dados gerenciados (Azure SQL e MongoDB Atlas) com backups automáticos. Separação clara entre dados de usuários (SQL) e transações (MongoDB). Processamento assíncrono via functions evita sobrecarga nos microserviços.
15. Disponibilidade: Functions serverless com alta disponibilidade e auto-scaling. CDN para frontend garante baixa latência global. API Gateway como ponto único de entrada com health checks.
16. Segurança: Comunicação HTTPS entre todos os componentes. JWT para autenticação stateless. Credenciais em variáveis de ambiente (secrets management). Bancos de dados com acesso restrito por IP/VNet.
17. Integração: APIs externas (Binance) acessadas de forma desacoplada através das functions. Event-driven architecture permite adicionar novos consumidores de eventos sem modificar o BFF.

7.2 Nível de Infraestrutura 2 – Detalhamento

Node 1: Frontend Hosting

- Hospeda o Shell App.
- Hospeda o microfrontend Dashboard que é carregado via iframe no Shell App.

Node 2: Backend Cluster (Docker)

- Container whale-usuarios.
- Container whale-portfolios.
- Container whale-bff.

Node 3: Serverless Functions

- Function Top 12 Cryptos
- Function para Transação

Node 4: Database Server

- SqlServer para usuários
- MongoDB para transações e portfolio

Serviços Externos

- API Binance → acessada pelo Serviço de Transações.
- Serviço de Notificação (SMTP/Push) → acessado pelo Serviço de Notificações.

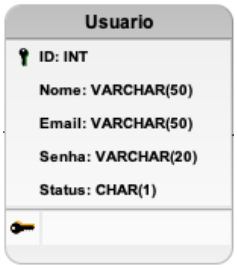

Diagram 11 - Diagrama Entidade Relacionamento - Banco de Dados de Usuários

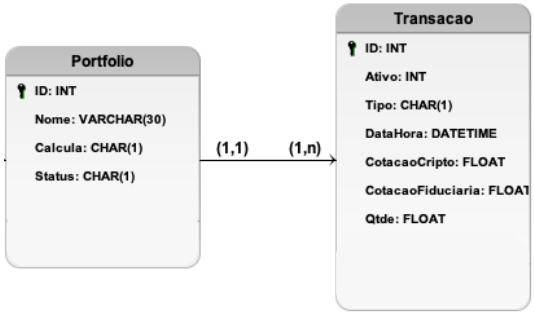

Diagram 12 - Diagrama não relacional - Banco de Dados Portfolio

8. Conceitos Transversais

Esta seção descreve conceitos, regras e soluções que se aplicam de forma transversal a múltiplas partes do sistema Whale, garantindo consistência, qualidade e manutenibilidade da arquitetura.

8.1 Interface do Usuário

Para a interface de usuário será utilizado um frontend em React com Vite como builder e Tailwind CSS para estilização, seguindo princípios de design responsivo e garantindo acesso fluido em diferentes dispositivos.

Design System:

- Glassmorphism: Todos os componentes utilizam backdrop-filter com blur e transparência para criar efeito de vidro fosco.
- Gradiente Animado: Background com animação suave de gradiente em tons de azul (#0f2027 → #203a43 → #2c5364), criando uma experiência visual moderna e atrativa.
- Modo Claro/Escuro: Sistema de tema implementado via Context API do React com persistência em localStorage.

8.2 Persistência de Dados

A persistência dos dados do sistema é realizada através de dois bancos de dados distintos, cada um otimizado para seu caso de uso específico:

8.2.1 Azure SQL Server (Usuários e Autenticação)

Tecnologia: Azure SQL Database com ADO.NET ou Entity Framework Core

Responsabilidade: Armazenar dados críticos de usuários e credenciais que requerem consistência ACID.

Padrões adotados:

- Repository Pattern: Abstração do acesso a dados através de interfaces.
- Unit of Work: Controle transacional de operações relacionadas.
- Connection Pooling: Reutilização de conexões para melhor performance.
- Parameterized Queries: Prevenção de SQL Injection.

8.2.2 MongoDB Atlas (Portfolios e Transações)

Tecnologia: MongoDB Atlas com MongoDB.Driver (.NET) ou Mongoose (Node.js na Function)

Responsabilidade: Armazenar documentos de portfolios e transações que se beneficiam de schema flexível e alta performance de leitura.

Padrões adotados:

- Document Model: Documentos ricos com subdocumentos embutidos quando apropriado
- Repository Pattern: Consistência com o padrão usado no SQL
- Indexes: Índices estratégicos para otimizar queries frequentes
- Connection Pooling: Pool de conexões configurado para alta concorrência

8.2.3 Modo Anônimo (localStorage)

Para usuários não autenticados, os dados são armazenados localmente no navegador usando localStorage.

8.3 Integração com APIs Externas

A atualização dos preços das criptomoedas é um requisito fundamental e será realizada por meio de consultas periódicas (polling) a API de mercado externa, o da Binance. Este mecanismo será a abordagem padronizada para buscar dados externos que necessitam de atualização constante, garantindo que o sistema opere com informações precisas e com atualizações a cada 30 segundos.

8.4 Padronização de Ambientes

Para garantir a portabilidade e a facilidade de implantação, o Whale utilizará Docker tanto no ambiente de desenvolvimento quanto para a distribuição final da aplicação. A

conteinerização padroniza o ambiente de execução, eliminando inconsistências entre diferentes máquinas

8.5 Injeção de Dependência para Modularidade

A arquitetura do backend em .NET utilizará o padrão de Injeção de Dependência para gerenciar as dependências entre os componentes. Essa abordagem promove um baixo acoplamento, facilita a testabilidade e a manutenção do sistema, permitindo que as implementações de serviços e repositórios sejam facilmente substituídas ou atualizadas sem impactar outras partes do código.

9. Decisões Arquiteturais

Esta seção documenta as principais decisões arquiteturais (ADRs - Architecture Decision Records) tomadas no projeto Whale, incluindo o contexto, alternativas consideradas, decisão final e suas consequências.

DA-01 — Arquitetura de Microserviços com BFF

Problema: Era necessário definir a arquitetura geral do sistema, garantindo escalabilidade, manutenibilidade e separação de responsabilidades.

Contexto: O sistema precisa gerenciar diferentes domínios (usuários, portfolios, transações) com possibilidade de escalar independentemente. O frontend precisa agregar dados de múltiplas fontes.

Restrições:

- Equipe pequena de desenvolvimento
- Tempo limitado de entrega
- Requisito de suportar modo anônimo e autenticado

Alternativas:

1. Monolito: Aplicação única com todos os módulos
2. Microserviços puros: Frontend comunica diretamente com cada microserviço
3. Microserviços com BFF: Camada intermediária (BFF) entre frontend e microserviçosDecisão: Foi adotado .NET/C# com uma API REST.

Consequências:

- Frontend mais simples, faz apenas 1 requisição ao invés de N
- Microserviços podem evoluir independentemente
- BFF permite adicionar cache e otimizações específicas para o frontend
- Facilita mudanças nos microserviços sem impactar o frontend
- Complexidade adicional de deploy (mais serviços para gerenciar)

- Latência adicional (BFF → Gateway → Microserviço)
- BFF pode se tornar ponto único de falha se não for replicado

---

DA-02 — Frontend em React com Microfrontend em iframe

Problema: Era preciso garantir que a interface fosse responsiva, modular e permitisse deploy independente de componentes.

Contexto: O dashboard de Top 12 Cryptos deve ser atualizado frequentemente e pode ser deployado independentemente da aplicação principal.

Restrições:

- A aplicação deve ser desenvolvida como SPA em React
- Suporte a modo anônimo com localStorage
- Design moderno com glassmorphism

Alternativas:

1. SPA Monolítica: Uma única aplicação React
2. Module Federation: Webpack 5 para compartilhar módulos
3. iframe: Carregar Dashboard MFE via iframe

Consequências:

- Deploy independente do Dashboard
- Isolamento total (CSS, JS, state)
- Simplicidade de implementação
- Facilita testes isolados do Dashboard
- Comunicação via postMessage é mais verbosa
- Duas aplicações React aumentam bundle size total
- iframe pode ter problemas de SEO (não crítico neste caso)
- Compartilhamento de autenticação requer postMessage ou cookies

---

DA-03 — Bancos de Dados: Azure SQL para Usuários, MongoDB para Portfolios

Problema: Era necessário escolher tecnologias de banco de dados adequadas para diferentes tipos de dados e padrões de acesso.

Contexto: Usuários e credenciais requerem consistência ACID e relacionamentos. Portfolios e transações têm estrutura flexível e alta frequência de escrita.

Restrições:

- Azure SQL Server Free (1 DTU) disponível
- MongoDB Atlas Free (M0) disponível
- Custos devem ser minimizados (usar tiers gratuitos)

Alternativas:

1. Tudo em SQL: Azure SQL para tudo
2. Tudo em NoSQL: MongoDB para tudo

Consequências:

- Melhor ferramenta para cada caso de uso
- SQL garante integridade de dados críticos (usuários)
- MongoDB oferece flexibilidade e performance para transações
- Separação reduz acoplamento entre domínios
- Equipe precisa conhecer dois bancos de dados
- Joins entre usuários e transações não são possíveis (agregação no BFF)
- Dois pontos de falha (dois bancos)
- Backup e restore mais complexos

DA-04 — Modo Anônimo com localStorage e Export/Import JSON

Problema: Era necessário permitir que usuários experimentem o sistema sem criar conta, mas ainda assim possam migrar dados posteriormente.

Contexto: Barreiras de cadastro reduzem conversão. Usuários querem testar antes de se comprometer. Dados precisam ser portáveis entre dispositivos.

Restrições:

- Sistema deve suportar uso sem autenticação
- Dados devem ser exportáveis em formato interoperável
- Importação deve validar integridade dos dados

Alternativas:

1. Obrigar cadastro: Sem modo anônimo
2. Sessão temporária no servidor: Backend armazena dados de sessões anônimas
3. localStorage + Export/Import: Dados no navegador com portabilidade via JSON
4. Decisão: Foi definido o uso de localStorage para armazenamento local com export/import de JSON.

Consequências:

- Zero fricção para novos usuários

- Privacidade (dados não saem do dispositivo)
- Portabilidade total via export/import
- Não requer backend para modo anônimo
- Facilita testes e demos
- Dados perdidos se limpar cache do navegador
- Não funciona em múltiplos dispositivos simultaneamente
- Import manual (usuário precisa baixar e fazer upload)
- Sem sincronização automática

10. Requisitos de qualidade

10.1 Árvore de qualidade

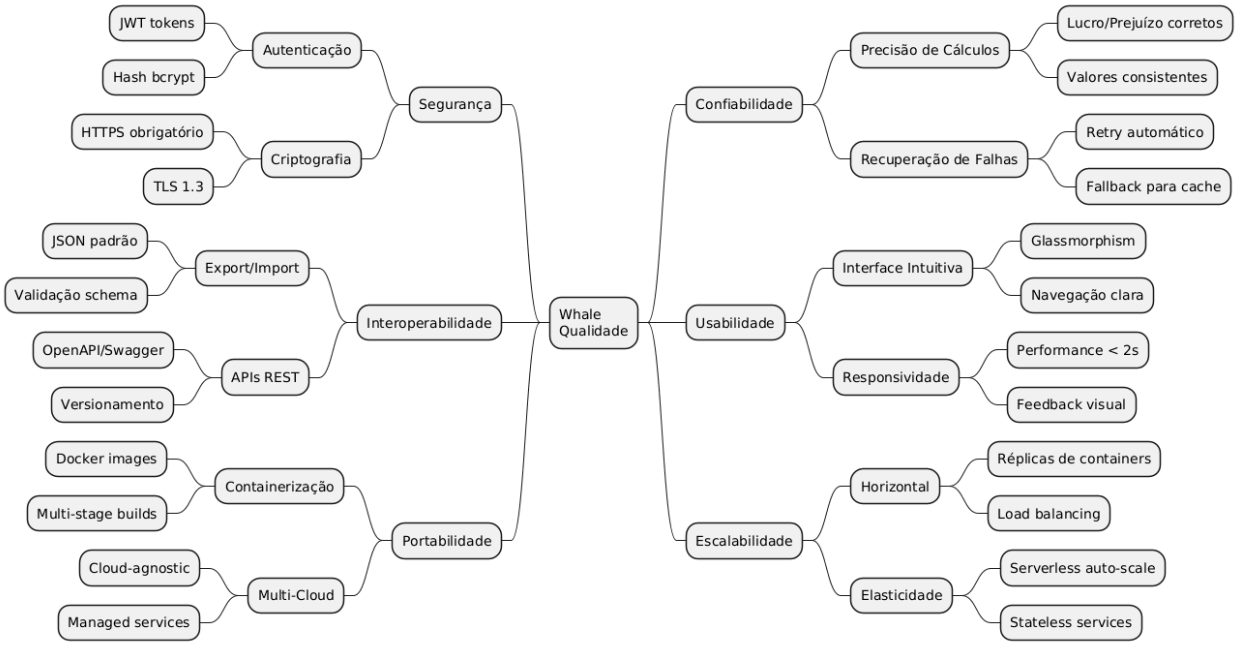

Diagrama 13 - Árvore de Qualidade

10.2 Cenários de Qualidade

Cenário 1: Disponibilidade com Falhas

Descrição: Mesmo com falhas de servidor, aplicação continua online.

Meta: Uptime 99.5%, detecção < 30s, recovery < 5min, zero downtime percebido.

Medição: Uptime Robot, logs de health checks.

Cenário 2: Pico de Acesso (10x usuários)

Descrição: Em grandes eventos, todos usuários acessam informações corretamente.

Meta: Suportar 1000 users simultâneos, P95 < 3s, erro < 1%, auto-scale < 60s.

Medição: Testes de carga (k6), APM.

Cenário 3: Cotações Atualizadas

Descrição: Usuário vê valores atualizados a cada 30 segundos.

Meta: Polling 30s, latência máxima 30s, cache hit > 80%, response < 500ms.

Medição: Logs de polling, métricas de cache.

Cenário 4: Proteção com Criptografia

Descrição: Dados protegidos em caso de vazamento.

Meta: bcrypt work factor ≥ 10, TDE 100%, HTTPS/TLS 1.3 100%, zero vazamentos.

Medição: Auditorias de segurança, pentests trimestrais.

Cenário 5: Backup de Dados

Descrição: Arquivos corrompidos substituídos por backup.

Meta: Backup Azure SQL automático diário (retenção 7d), RPO < 24h, RTO < 1h.

Medição: Testes de restore mensais.

Cenário 6: Notificações de Variação

Descrição: Usuário notificado instantaneamente em grandes variações de preço.

Meta: Detecção de variação ≥ 5%, latência notificação < 10s, taxa entrega > 95%.

Medição: Logs de notificações, timestamps.

Cenário 7: Performance de Transações (adicional)

Descrição: Adicionar transação deve ser rápido e confiável.

Meta: Response < 2s (202 Accepted), precisão cálculos 100%, processamento async < 10s.

Medição: APM tracing, logs de processamento.

11. Riscos e Débitos Técnicos

11.1 Riscos Identificados

- Dependência de Serviços Externos (Binance): O sistema depende de APIs públicas/privadas de exchanges para obter cotações. Interrupções ou mudanças de contrato nessas APIs podem afetar a disponibilidade do serviço. Mitigação: APIs alternativas: Integrar CoinGecko API como fallback secundário; Integrar CoinMarketCap API como fallback terciário; Switch automático se Binance falhar por > 5 minutos.
- Ponto Único de Falha no API Gateway: Como todas as requisições passam pelo API Gateway, sua indisponibilidade compromete o acesso a todos os serviços backend. Mitigação: Alta disponibilidade com múltiplas réplicas: Alta disponibilidade com múltiplas réplicas, com no mínimo 2 réplicas de BFF e Gateway em produção.
- Limite de Requisições na Binance API: O excesso de consultas pode atingir limites de taxa (rate limit), causando falhas no registro de transações e na atualização de preços. Mitigação: Cache compartilhado entre todas as instances, todas as functions consultam o cache antes de chamar Binance.
- Cold Start das Serverless Functions: Serverless functions têm cold start delay (~2-5 segundos) quando não são invocadas por alguns minutos. Isso pode causar latência perceptível para o primeiro usuário após período de inatividade. Mitigação: Warm-up requests, chamando função a cada 5 minutos.

11.2 Dívidas Técnicas

- Monitoramento Incompleto: Falta de informações sobre o estado atual da aplicação.
- Testes Limitados de Alta Carga e Falha: Faltam testes sistemáticos de estresse (load testing) e simulações de falhas regionais (chaos engineering) para validar a resiliência da solução.
- Pipeline de CI/CD Manual em Partes: Algumas etapas de deploy ainda exigem intervenção manual, o que aumenta riscos de erro humano e reduz a velocidade de entrega.
- Falta de API de Cotação alternativas: O sistema depende exclusivamente da Binance API. Se a Binance ficar fora do ar ou bloquear o acesso, não há fallback para outras fontes de cotação.

12. Glossário

Termo	Definição
Portfólio	Conjunto de criptomoedas de um investidor.
Polling	Técnica de consultar uma API em intervalos regulares para buscar atualizações.
Ativo	Uma criptomoeda específica registrada no sistema.
API Gateway	Componente que centraliza e gerencia chamadas a serviços backend, realizando autenticação, roteamento e controle de acesso.
API REST	Representational State Transfer: Padrão arquitetural que permite a comunicação entre sistemas via HTTP, geralmente com dados em formato JSON.
CI/CD	Continuous Integration / Continuous Delivery: Práticas de integração e entrega contínua que automatizam testes, builds e deploys de software.
Container Docker	Unidade leve de software que empacota código e dependências para garantir que a aplicação rode de forma consistente em diferentes ambientes.
JWT	SON Web Token: Padrão aberto de token usado para autenticação e troca segura de informações entre partes.
Repositório	Camada de software responsável por acessar e manipular dados no banco de forma estruturada.
Push Notification	Mensagem enviada automaticamente para o dispositivo de um usuário, mesmo quando ele não está usando ativamente a aplicação.