# UBS WATCHDOG

**GRUPO 2**
Documentação do Projeto
Processo Seletivo Trainee 2026

# Sumário

## Controle de versão

| Versão | Data | Autor | Mudanças |
|---|---|---|---|
| 1.0 | 06/01/2026 | Maria Eduarda | Criação do esqueleto inicial da documentação. |
| 2.0 | 08/01/2026 | Maria Eduarda | Preenchimento dos itens 1, 2, 3 e 4. |
| 3.0 | 13/01/2026 | Maria Eduarda | Preenchimento do item de Front-end. |
| 4.0 | 14/01/2026 | Gabriel Santana | Preenchimento de Status do Projeto |
| 5.0 | 14/01/2026 | João Rudge | Preenchimento das pendências de Backend |
| 6.0 | 14/01/2026 | Fellipe | Preenchimento de Auth e Segurança |
| 7.0 | 14/01/2026 | Thales | Preenchimento do item de Testes |

# 1. Visão geral

## 1.1. Contexto e objetivo

Sistema para monitoramento de transações financeiras e compliance: registro de clientes e transações, aplicação de regras, geração de alertas e relatórios.

## 1.2. Escopo

- Cadastro e consulta de clientes
- Registro e consulta de transações
- Regras automáticas de compliance para identificar transações suspeitas
- Geração/consulta de alertas para analistas
- Relatórios e painéis para acompanhamento

## 1.3. Público-alvo

- Analista de risco/compliance
- Gestores
- Equipe técnica

# 2. Equipe

## 2.1. Equipe e responsabilidades

| Nome | Equipe | Responsabilidade | GitHub |
|---|---|---|---|
| Gabriel Candido Santana | Back | Líder da Equipe | GabrielSantana003 |
| Fellipe Tripovichy Andrade | Front | Líder de Front | Fellipe-Tripovichy |
| João Augusto de Andrade Malta Rudge | Back | Líder de Back | RudgeJoao |
| Maria Eduarda Ribeiro Facio | Front | Líder de Documentação | dudaribeiro7 |
| Thales Nogueira | Front | Líder de QA | Thalessns |

## 2.2. Rituais de comunicação

- Cadência de reuniões: 1 vez por semana
- Canal principal: WhatsApp e Discord
- Kanban / gestão de tarefas: https://www.notion.so/839510d4e20541d887b8858685538c1f?v=de99277270c844e28a944dc75cf36f81

# 3. Requisitos

## 3.1 Requisitos funcionais

| ID | Descrição | Prioridade | Status |
|---|---|---|---|
| RF-01 | Autenticar usuário no front (login mock com usuário/senha) para liberar acesso às telas | MVP | Concluído |
| RF-02 | Cadastrar cliente com campos: Id, Nome, País, Nível de Risco, KYC Status | MVP | Concluído |
| RF-03 | Listar clientes com filtros básicos e acessar detalhes de um cliente | MVP | Concluído |
| RF-04 | Registrar transação (Depósito/Saque/Transferência) com campos: Id, ClienteId, Tipo, Valor, Moeda, Contraparte, DataHora | MVP | Concluído |
| RF-05 | Consultar histórico de transações por cliente com filtro de período | MVP | Concluído |
| RF-06 | Executar regras de compliance automaticamente | MVP | Concluído |
| RF-07 | Criar alerta automático quando uma regra de compliance for violada (Id, ClienteId, TransacaoId, Regra, Severidade, Status) | MVP | Concluído |
| RF-08 | Listar alertas com filtros por Status e Severidade | MVP | Concluído |
| RF-09 | Atualizar status do alerta (Novo → Em Análise → Resolvido) | MVP | Concluído |
| RF-10 | Gerar relatório por cliente no período: total movimentado + número de alertas | MVP | Concluído |
| RF-11 | Exibir relatório no front com gráfico + tabela | Should | Concluído |

## 3.2 Requisitos não funcionais

- **Segurança**: autenticação com usuário e senha, chamadas de API protegidas.
- **Usabilidade**: UI responsiva, estados de loading/erro, filtros simples e navegação via React Router.

## 3.3 Regras de negócio (compliance)

| ID | Descrição da regra |
|---|---|
| RN-01 | Limite diário: soma das transações do cliente no dia não pode ultrapassar um limite configurado |
| RN-02 | Transferência internacional para país de risco: se Tipo = Transferência e país da contraparte estiver na lista de risco → alerta |
| RN-03 | Structuring (fracionamento): várias transações pequenas no mesmo dia |

# 4. Arquitetura geral

## 4.1. Visão geral

Frontend consumindo API REST com persistência em PostgreSQL.

## 4.2. Tecnologias

| Camada | Tecnologia | Observações |
|---|---|---|
| Frontend | Next.js + React + TypeScript + Tailwind | Atomic Design; deploy em Vercel. |
| Backend | .NET 8 (C#) | DDD |
| Banco de Dados | PostgreSQL | EF Core + migrations. |
| Auth | Firebase Authentication | Validação no backend. |
| Documentação | DOCX + Swagger + Kanban | Links em Anexos. |

# 5. Front-end

## 5.1 Telas

- Autenticação: login com usuário e senha.
- Clientes: lista com filtros básicos e formulário de cadastro.
- Transações: cadastro e histórico por cliente.
- Compliance/Alertas: listagem com filtros por status/severidade e opção de resolução/atualização.
- Relatórios: resumo por cliente (gráfico de barras/pizza + tabela).

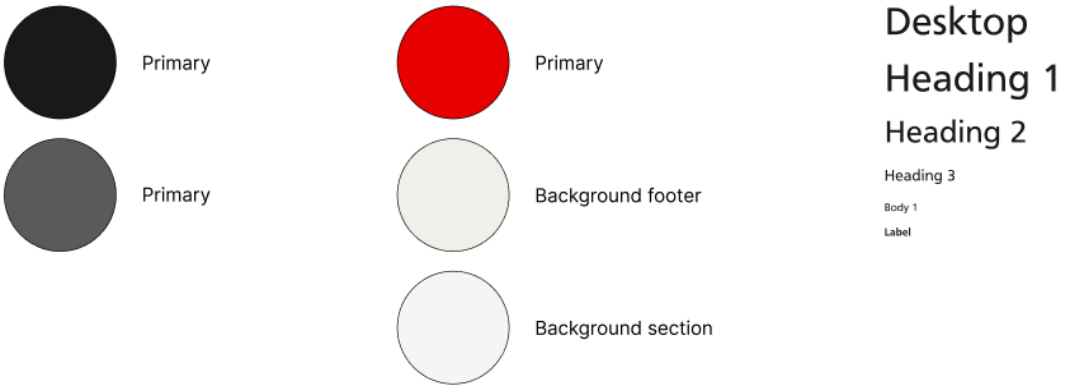
## 5.2 Stack

Principais tecnologias: Next.js, React, TypeScript, Tailwind, Redux Toolkit, Firebase.

## 5.3 Design system e Atomic Design

O design system do projeto foi definido a partir do **padrão visual do UBS**, usando como referência direta o **site oficial** para garantir consistência com a identidade da empresa (cores, tipografia, espaçamentos e comportamento de componentes).

- **Cores:** paleta baseada nas cores institucionais do UBS (primária, backgrounds e neutros).
- **Tipografia:** hierarquia de títulos e textos (Heading 1–3, Body, Label), com pesos e tamanhos padronizados.
- **Espaçamentos:** escala de spacing consistente (margens, paddings e gaps) aplicada em toda a UI.
- **Bordas e raios:** padrão único de arredondamento e separação visual.
- **Estados:** hover, focus, disabled, loading e validação (erro/sucesso) definidos por componente.



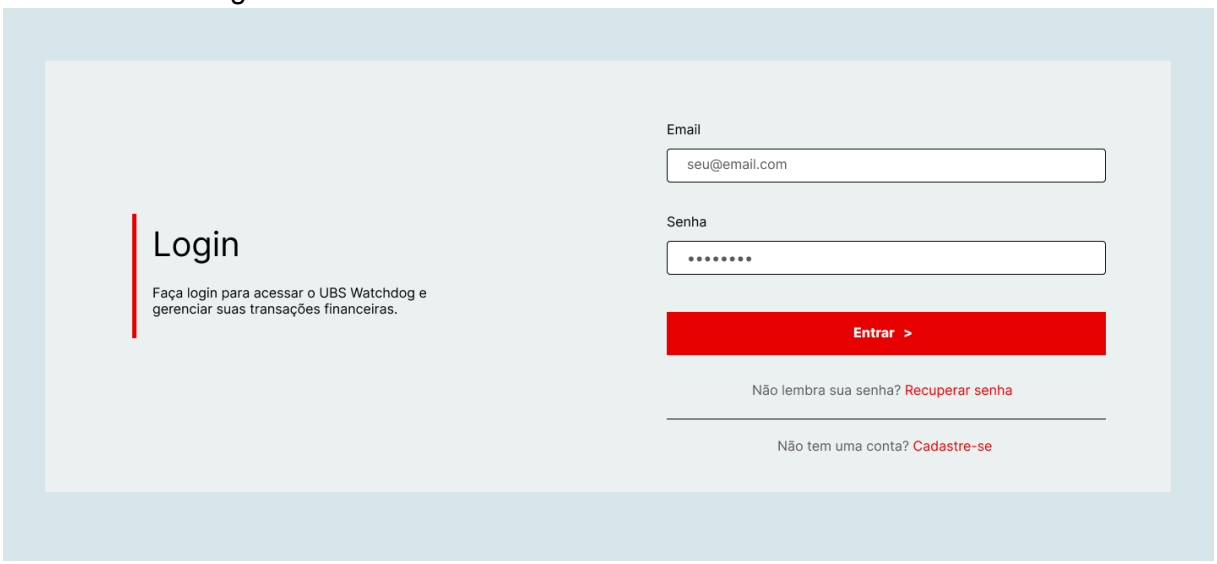

A partir dessa base, implementamos os componentes no front-end em **React + TypeScript**, utilizando o **shadcn/ui** para acelerar a construção e manter padronização e reutilização. Entre os componentes mais importantes implementados estão os de base de UI (Button, Input, Select, Card, Badge, Tooltip, Popover), navegação e estrutura (NavigationBar,

Breadcrumb, LayoutWrapper, Footer, SectionTitle e HeroTitle), filtros e datas (Calendar e DatePickerInput), exibição de dados (Table, DataTable, CardTable e Pagination), feedback/estados (Spinner e Empty), além de componentes específicos do produto como ComplianceCard, ReportCard, TransactionCard e ResumeTransactionCard, e visualizações com gráficos (BarChart e PieChart). Também existe a camada de inicialização de autenticação via AuthInitializer, garantindo o fluxo de acesso do sistema.

## 5.4 Wireframes

Os wireframes do projeto foram produzidos integralmente no Figma para definir a estrutura das telas, hierarquia de informação e fluxo de navegação antes da implementação. Foram desenhadas as telas de Login, Relatórios, Listagem de Clientes, Transações e Alertas, já considerando os principais componentes do design system (cards, tabelas, filtros e paginação) e os estados básicos de uso (carregamento e ausência de dados). Esses wireframes serviram como guia direto para o desenvolvimento em React/TypeScript, garantindo consistência visual e reduzindo retrabalho durante a implementação.

Tela de Login:



Tela de Clientes:

## Relatórios

Visão geral da carteira. Selecione um cliente para detalhar perfil, transações e alertas.

Data inicial

Data final

| Cliente | País | Nível de risco | Status KYC | Transações | Alertas | Ações |
|---|---|---|---|---|---|---|
| Fulano de tal | Brasil | Baixo | Aprovado | 5 | 1 | ↗ |
| Fulano de tal | Brasil | Baixo | Aprovado | 5 | 1 | ↗ |
| Fulano de tal | Brasil | Baixo | Aprovado | 5 | 1 | ↗ |

Tela de Relatórios:

## Relatório do cliente

Visão geral da carteira. Selecione um cliente para detalhar perfil, transações e alertas.

Moeda

Data inicial

Data final

### Informações gerais

Detalhes do cliente:
ID
Nacionalidade
Nível de Risco
KYC Status

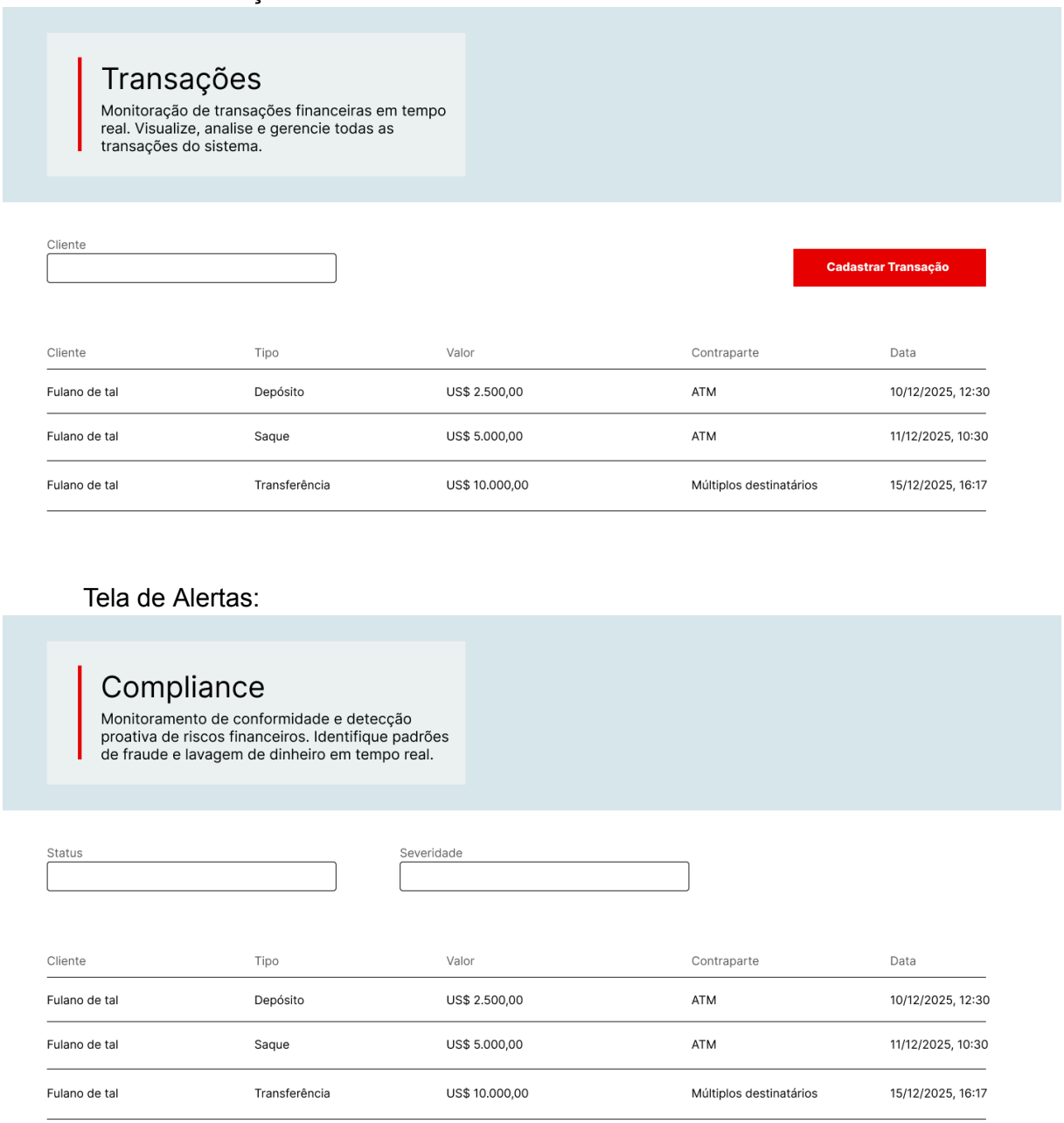
Total movimentado no período na moeda selecionada

Número de alertas ativo no período

Gráfico de pizza de alertas por severidade e gráfico de barras de tipos de transações

Tabelas com detalhes de transações e alertas

Tela de Transações:




Tela de Alertas:




## 5.5 Como rodar (dev e deploy)

Para rodar o projeto em desenvolvimento (dev), instale as dependências e suba o servidor local. No diretório do projeto, execute **npm install** (ou yarn, pnpm install, bun install) e depois inicie com **npm run dev** (ou yarn dev, pnpm dev, bun dev). A aplicação ficará disponível em **http://localhost:3000** e atualiza automaticamente conforme você edita os arquivos.

No deploy, a aplicação está publicada na Vercel e pode ser acessada em https://ubs-watchdog-frontend-group2.vercel.app/.

# 6. Back-end

## 6.1 Arquitetura (Clean Architecture)

O projeto segue os princípios da **Clean Architecture**, organizado em 4 camadas principais:

```
UBS.Watchdog
├── Domain # Camada de Domínio (núcleo)
├── Application # Camada de Aplicação (casos de uso)
├── Infrastructure # Camada de Infraestrutura (persistência)
└── API # Camada de Apresentação (Web API)
```

**Domain (Domínio)**

- **Responsabilidade**: Contém as regras de negócio e entidades do domínio
- **Componentes**:
  - **Entities**: `Cliente`, `Transacao`, `Alerta`
  - **Enums**: `NivelRisco`, `StatusKyc`, `TipoTransacao`, `Moeda`, `SeveridadeAlerta`, `StatusAlerta`
  - **Value Objects**: `Contraparte`
- **Características**:
  - Independente de frameworks e tecnologias
  - Contém lógica de negócio pura
  - Não possui dependências externas

**Application (Aplicação)**

- **Responsabilidade**: Orquestra o fluxo de dados e implementa casos de uso
- **Componentes**:
  - **Services**: `ClienteService`, `TransacaoService`, `AlertaService`, `ReportService`, `ComplianceService`
  - **DTOs (Data Transfer Objects)**: Request/Response models
  - **Mappings**: Conversão entre Entities e DTOs
  - **Compliance (Regras de Negócio)**:
    - Interface `IRegraCompliance` (Strategy Pattern)
    - `RegraLimiteDiario`: Valida limite diário de transações
    - `RegraPaisAltoRisco`: Detecta transferências para países de risco
- **Características**:
  - Depende apenas da camada Domain
  - Implementa casos de uso do sistema
  - Orquestra validações de compliance

**Infrastructure (Infraestrutura)**

- **Responsabilidade**: Implementa acesso a dados e serviços externos

- **Componentes**:
  - **Data**: `AppDbContext`, `DbSeeder`
  - **Configurations**: Entity Framework configurations
  - **Repositories**: Implementação dos repositórios
    - `ClienteRepository`
    - `TransacaoRepository`
    - `AlertaRepository`
  - **Migrations**: Versionamento do banco de dados
- **Características**:
  - Implementa Repository Pattern
  - Entity Framework Core para ORM
  - PostgreSQL como banco de dados

**API (Apresentação)**

- **Responsabilidade**: Exposição dos endpoints REST
- **Componentes**:
  - **Controllers**:
    - `ClientesController`: CRUD de clientes + atualização KYC/Risco
    - `TransacoesController`: Registro e consulta de transações
    - `AlertasController`: Gestão de alertas de compliance
    - `RelatoriosController`: Geração de relatórios
    - `HealthController`: Health check da aplicação
  - **Program.cs**: Configuração da aplicação (DI, Middleware, CORS)

## 6.2 Banco de dados e migrations

### Modelo Relacional

O banco de dados possui 3 tabelas principais com os seguintes relacionamentos:

**Tabela: Clientes**

- Armazena informações dos clientes do sistema
- Um cliente pode ter múltiplas transações (relação 1:N)
- Um cliente pode ter múltiplos alertas (relação 1:N)

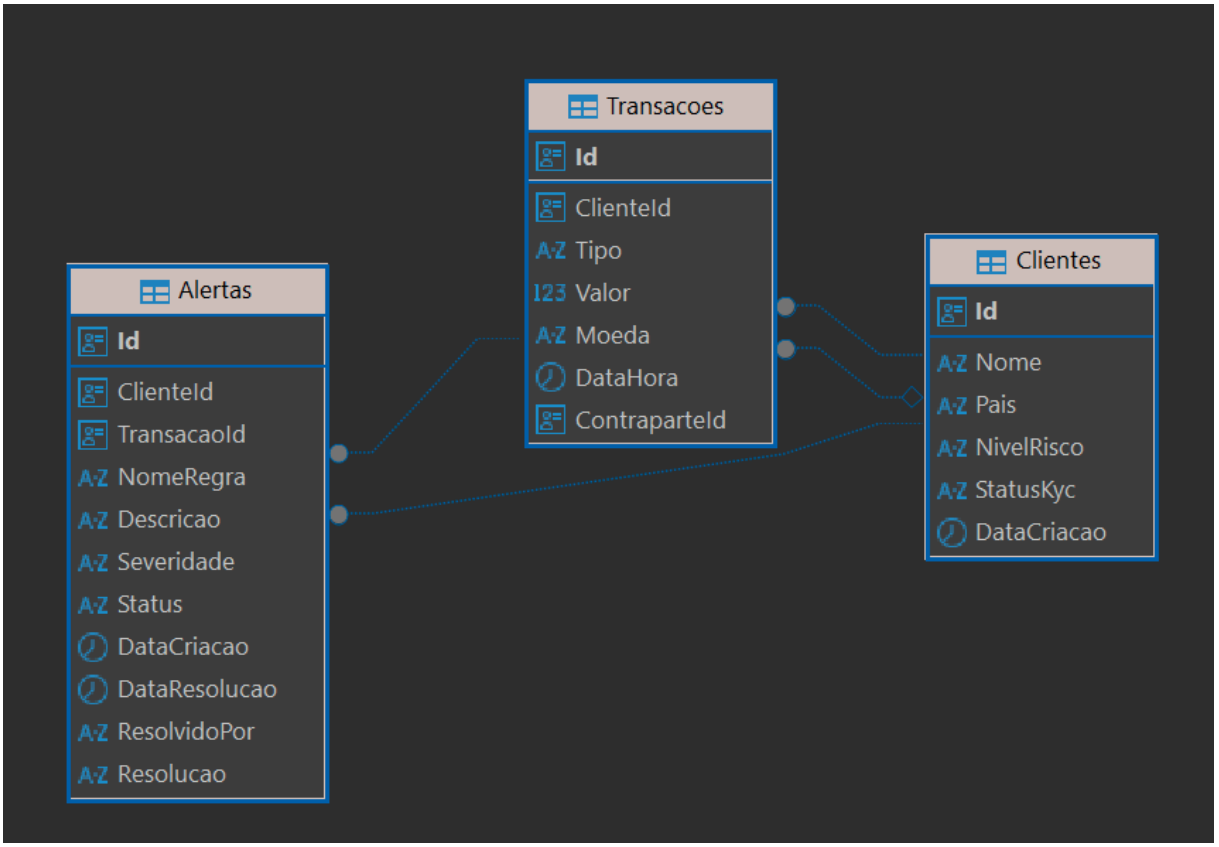
**Tabela: Transacoes**

- Registra todas as operações financeiras
- Cada transação pertence a um cliente (FK: ClienteId)
- Uma transação pode gerar múltiplos alertas (relação 1:N)

**Tabela: Alertas**

- Armazena violações de regras de compliance
- Cada alerta pertence a um cliente (FK: ClienteId)
- Cada alerta está relacionado a uma transação específica (FK: TransacaoId)

**Relacionamentos:**




- `Transacoes.ClienteId` → `Clientes.Id`
- `Alertas.ClienteId` → `Clientes.Id`
- `Alertas.TransacaoId` → `Transacoes.Id`

## 6.3 Endpoints (Swagger)

Link do Swagger: http://72.62.141.100:6001/swagger/index.html

### Clientes (`/api/clientes`)

| Método | Endpoint | Descrição |
|---|---|---|
| `POST` | `/api/clientes` | Criar novo cliente no sistema |
| `GET` | `/api/clientes` | Listar todos os clientes cadastrados |
| `GET` | `/api/clientes/{id}` | Buscar cliente específico por ID (UUID) |
| `GET` | `/api/clientes/pais/{pais}` | Listar clientes filtrados por país |

| | | |
|---|---|---|
| GET | /api/clientes/nivel-risco/{nivel} | Listar clientes por nível de risco (Baixo, Medio, Alto) |
| PATCH | /api/clientes/{id}/kyc | Atualizar status KYC do cliente (Pendente, Aprovado, Rejeitado) |
| PATCH | /api/clientes/{id}/nivel-risco | Atualizar nível de risco do cliente (Baixo, Medio, Alto) |
| DELETE | /api/clientes/{id} | Remover cliente do sistema (soft delete) |

**Request Body - POST /api/clientes**

```
{
  "nome": "string (obrigatório)",
  "pais": "string (obrigatório)",
  "nivelRisco": "Baixo | Medio | Alto (obrigatório)"
}
```

**Response - ClienteResponse**

```
{
  "id": "uuid",
  "nome": "string",
  "pais": "string",
  "nivelRisco": "Baixo | Medio | Alto",
  "statusKyc": "Pendente | Aprovado | Rejeitado",
  "dataCriacao": "datetime"
}
```

## Transações (/api/transacoes)

| Método | Endpoint | Descrição |
|---|---|---|
| POST | /api/transacoes | Registrar nova transação (executa validação de compliance automaticamente) |
| GET | /api/transacoes | Listar todas as transações do sistema |
| GET | /api/transacoes/{transacaoId} | Buscar transação específica por ID |
| GET | /api/transacoes/clientes/{clienteId} | Listar todas as transações de um cliente (com filtro opcional de período) |
| GET | /api/transacoes/filtrar | Filtrar transações por múltiplos critérios (clienteId, período, moeda, tipo) |

**Request Body - POST /api/transacoes**

```
{
  "clienteId": "uuid (obrigatório)",
  "tipo": "Deposito | Saque | Transferencia (obrigatório)",
  "valor": "decimal > 0.01 (obrigatório)",
  "moeda": "USD | BRL | EUR (obrigatório)",
  "contraparteId": "uuid (opcional - obrigatório para Transferencia)"
}
```

**Query Parameters - GET /api/transacoes/clientes/{clienteId}**

- `dataInicio` (opcional): Data inicial do período (ISO 8601)
- `dataFim` (opcional): Data final do período (ISO 8601)

**Query Parameters - GET /api/transacoes/filtrar**

- `clienteId` (opcional): UUID do cliente
- `dataInicio` (opcional): Data inicial
- `dataFim` (opcional): Data final
- `moeda` (opcional): USD, BRL ou EUR
- `tipo` (opcional): Deposito, Saque ou Transferencia

## Alertas (`/api/alertas`)

| Método | Endpoint | Descrição |
|---|---|---|
| GET | /api/alertas | Listar todos os alertas do sistema |
| GET | /api/alertas/{id} | Buscar alerta específico por ID |
| GET | /api/alertas/cliente/{clienteId} | Listar todos os alertas de um cliente específico |
| GET | /api/alertas/status/{status} | Filtrar alertas por status (Novo, EmAnalise, Resolvido) |
| GET | /api/alertas/filtrar | Filtrar alertas por múltiplos critérios avançados |
| PATCH | /api/alertas/{id}/iniciar-analise | Mover alerta de "Novo" para "Em Análise" |
| PATCH | /api/alertas/{id}/resolver | Resolver alerta (mover para status "Resolvido") |

**Query Parameters - GET /api/alertas/filtrar**

- `status` (opcional): Novo, EmAnalise, Resolvido
- `severidade` (opcional): Baixa, Media, Alta, Critica
- `clienteId` (opcional): UUID do cliente
- `dataCriacaoInicio` (opcional): Data inicial de criação
- `dataCriacaoFim` (opcional): Data final de criação
- `dataResolucao` (opcional): Data de resolução

**Request Body - PATCH /api/alertas/{id}/resolver**

```
{
  "resolvidoPor": "string (obrigatório)",
  "resolucao": "string (obrigatório)"
}
```

**Response - AlertaResponse**

```
{
  "id": "uuid",
  "clienteId": "uuid",
  "transacaoId": "uuid",
  "nomeRegra": "string",
  "descricao": "string",
  "severidade": "Baixa | Media | Alta | Critica",
  "status": "Novo | EmAnalise | Resolvido",
  "dataCriacao": "datetime",
  "dataResolucao": "datetime?",
  "resolucao": "string?",
  "resolvidoPor": "string?"
}
```

## Relatórios (`/api/relatorios`)

| Método | Endpoint | Descrição |
|---|---|---|
| GET | `/api/relatorios/cliente/{id}` | Gerar relatório completo de um cliente (transações, alertas, estatísticas) |
| GET | `/api/relatorios/filtrar` | Gerar relatório filtrado por múltiplos critérios |

**Query Parameters - GET /api/relatorios/cliente/{id}**

- `dataInicio` (opcional): Data inicial do período
- `dataFim` (opcional): Data final do período

**Query Parameters - GET /api/relatorios/filtrar**

- `statusAlerta` (opcional): Filtrar por status de alertas
- `statusKyc` (opcional): Filtrar por status KYC

- `pais` (opcional): Filtrar por país

**Response - RelatorioClienteResponse**

```
{
  "clienteId": "uuid",
  "nomeCliente": "string",
  "pais": "string",
  "nivelRisco": "string",
  "statusKyc": "string",
  "dataCriacao": "datetime",
  "totalTransacoes": "int",
  "totalMovimentado": "decimal",
  "mediaTransacao": "decimal",
  "dataUltimaTransacao": "datetime?",
  "totalAlertas": "int",
  "alertasNovos": "int",
  "alertasEmAnalise": "int",
  "alertasResolvidos": "int",
  "alertasCriticos": "int",
  "periodoInicio": "datetime?",
  "periodoFim": "datetime?"
}
```

**Enums Disponíveis:**

- **NivelRisco**: Baixo, Medio, Alto
- **StatusKyc**: Pendente, Aprovado, Rejeitado
- **TipoTransacao**: Deposito, Saque, Transferencia
- **Moeda**: USD, BRL, EUR
- **SeveridadeAlerta**: Baixa, Media, Alta, Critica
- **StatusAlerta**: Novo, EmAnalise, Resolvido

# 7. Integração Front x Back

## 7.1. Contrato de API

### Padrões adotados

- URLs seguem padrão REST (/api/recurso)
- Verbos HTTP semânticos (GET, POST, PATCH, DELETE)
- Respostas padronizadas com códigos HTTP

## 7.2. Auth e segurança

O projeto utiliza **Firebase Authentication** como provedor de identidade no **Front-end**, responsável pelo processo de login e gerenciamento de usuários.

- O Front-end realiza autenticação diretamente com o Firebase
- O Firebase retorna um **ID Token (JWT)**
- O token representa a identidade do usuário autenticado

Para o escopo do projeto:

- O Back-end não gerencia credenciais diretamente
- O token do Firebase pode ser enviado nas requisições HTTP via header Authorization
- A API está preparada para validar tokens futuramente

Segurança

- Autenticação delegada ao Firebase reduz exposição de credenciais
- Identificadores do sistema utilizam GUIDs
- Regras de negócio e validações são executadas no Back-end
- Logs estruturados permitem auditoria e rastreabilidade

# 8. Testes e qualidade

## 8.1. Estratégia de testes

**Backend:**

- **Testes de Integração com XUnit:** validam o funcionamento completo dos endpoints e sua interação com o banco de dados real.
- **Validação:** Testa desde a persistência de dados até a resposta HTTP.
- **Banco de Dados Isolado:** Cada teste usa um banco em container isolado para evitar interferência.

**Frontend:**

- **Testes unitários e de componentes:** validação de regras de tela, renderização e comportamentos principais (ex.: formulários, filtros, listagens).
- **Validação automática no fluxo de PR:** a cada pull request, é executado um pipeline que checa **build**, **qualidade do código** e **testes** antes do merge.
- **Análise de qualidade com Sonar:** integração com o Sonar para revisar **bugs**, **vulnerabilidades**, **code smells** e manter padrão consistente no repositório.
- **Prevenção de regressões:** as validações garantem que novas alterações não quebrem funcionalidades existentes e mantenham estabilidade do sistema.

## 8.2. Critérios de qualidade

**Backend:**

- Todos os endpoints (GET, POST, PUT, DELETE) foram testados.
- Todos os status code possíveis (códigos 200, 400, 404, 500).
- Validação de payloads e respostas.
- Cobertura mínima de código de 80%.

**Frontend:**

- **Componentes e telas principais testados** (ex.: login, listagens, relatórios, filtros e formulários).
- **Validação de fluxos críticos** (ex.: submissão de formulário, busca/seleção de cliente, filtro por data).
- **Qualidade garantida pelo Sonar**: sem issues críticas de **bugs**, **vulnerabilidades** e **code smells** no merge.
- **Build e validações obrigatórias em PR**: merge só após passar em **testes + checagens de qualidade**.
- **Cobertura mínima de testes** definida para o projeto (ex.: **70% ou 80%**, conforme meta do time).

# 9. Status do projeto

## 9.1. Backlog e Kanban

Link do quadro Kanban:
https://www.notion.so/839510d4e20541d887b8858685538c1f?v=de99277270c844e28a944dc75cf36f81

Regras do fluxo: A Fazer → Em Andamento → Concluído

## 9.2. Riscos e decisões

Riscos:

| Risco | Impacto | Mitigação | Responsável |
|---|---|---|---|
| Escopo grande para o tempo disponível | Atraso na entrega | Priorizar MVP com features essenciais | Equipe |
| Falta de experiência em Domain-Driven Design | Código confuso | Estudo, estrutura clara e divisão de responsabilidades | Back-end |
| Integração Front-end e Back-end | Retrabalho | Contrato claro via Swagger | Equipe |

Decisões:

| Data | Decisão | Motivo | Impacto |
|---|---|---|---|
| 16/12/2025 | Utilização de arquitetura DDD | Facilitar manutenção e escalabilidade | Código organizado e separação clara de camadas |

# 10. Anexos

Links úteis:

- Repositório GitHub: https://github.com/Fellipe-Tripovichy/ubs-watchdog-frontend-group2
- Design system: https://www.figma.com/design/5HeLFhSgztquNfUIVNFKJL/UBS-Frontend-2?node-id=0-1
- Wireframes: https://www.figma.com/design/5HeLFhSgztquNfUIVNFKJL/UBS-Frontend-2?node-id=2-3
- Swagger/Contrato de API: http://72.62.141.100:6001/swagger/index.html
- Kanban: https://www.notion.so/839510d4e20541d887b8858685538c1f?v=de99277270c844e28a944dc75cf36f81