Documento de Requisitos: App de Gestão Financeira & Investimentos Inteligente

1. Visão Geral

Um sistema web pessoal para consolidar finanças diárias (contas e cartões) e carteira de investimentos (ações, FIIs, renda fixa) em um único local, utilizando automação via arquivos OFX e inteligência artificial para análise de dados.

---

2. Pilares Estratégicos

- **Esforço Manual Mínimo:** Automação via upload de OFX e IA.
- **Visão de Patrimônio Líquido:** Foco no crescimento real do patrimônio (Ativos - Passivos).
- **Privacidade e Custo Zero:** Uso de ferramentas open-source e camadas gratuitas (Tier Free).

---

3. Módulos e Funcionalidades

A. Dashboard Inicial (Torre de Comando)

- **Cards de KPI:**
    - **Patrimônio Total:** Valor consolidado (Investimentos + Saldos).
    - **Saldo Disponível:** Somatório de saldos em contas corrente.
    - **Fatura Acumulada:** Valor total de compras em cartões no mês atual.
- **Gráficos Rápidos:**
    - Barras: Receitas vs. Despesas do mês atual.
    - Pizza: Alocação de ativos (Ações, FIIs, Renda Fixa).
- **Feed Recente:** Últimas 5 transações importadas.

B. Gestão Financeira (Módulo OFX)

- **Multi-Contas:** Cadastro de diferentes instituições (Conta Gastos vs. Conta Investimento).
- **Importação OFX/CSV:** Upload de arquivos de conta corrente e faturas de cartão.
- **Motor de Deduplicação:** Filtro automático para transações já importadas (via ID único do OFX).
- **Lógica de Transferência Interna:** Identificação de fluxos entre contas próprias e pagamentos de faturas para evitar duplicidade de gastos.

- **Categorização Automática via IA:** Envio de descrições de faturas para IA classificar em categorias (Alimentação, Lazer, etc.) e com cache inteligente das informações.

C. Gestão de Investimentos

- **Lançamentos Manuais:** Cadastro de operações (Compra/Venda) com Ticker, Quantidade e Preço Médio.
- **Integração com Cotações:** Busca automática de preços atuais (API Brapi) para ativos da B3.
- **Cálculo de Rentabilidade:** Comparação entre o custo de aquisição e o valor de mercado atual.
- **Distribuição de Carteira:** Visualização por classe de ativos e setor.

D. Histórico de Patrimônio Líquido

- **Gráfico de Linha do Tempo:** Evolução mensal do patrimônio total.
- **Snapshots Mensais:** Registro automático do saldo no último dia do mês para comparação histórica.
- **Visão Ativos vs. Passivos:** Gráfico de área mostrando o crescimento dos bens contra as dívidas (faturas).

E. Camada de Inteligência Artificial (Gemini/Groq)

- **Analista Financeiro:** Botão para gerar relatório de saúde financeira com base nos gastos do mês.
- **Análise de Carteira:** Insights da IA sobre diversificação e exposição a riscos nos investimentos.
- **Chat de Dados:** Interface de chat para perguntas rápidas sobre o histórico financeiro.

---

4. Stack Tecnológica Sugerida

- **Frontend:** React + Tailwind CSS (Componentes Tremor para gráficos).
- **Backend:** Node.js.
- **Banco de Dados:** Supabase (PostgreSQL).
- **APIs Externas:**
    - **Brapi:** Cotações de ações e FIIs.
    - **Google AI Studio (Gemini):** Motor de IA gratuito.
- **Parser:** `node-ofx-parser` para leitura dos arquivos bancários.

---

5. Fluxo de Uso Ideal

1. O usuário faz download dos OFXs no banco.
2. Realiza o upload no App.
3. O App limpa duplicados e a IA categoriza os novos gastos.
4. O App atualiza o preço dos investimentos via API.
5. O Dashboard é atualizado com o novo **Patrimônio Líquido**.

Esquema de Banco de Dados:

Versão: 1.1

Foco: Automação, Deduplicação por Hash e Inteligência de Cache.

---

1. Gestão de Usuários e Contas

Estrutura fundamental para multi-usuário e multi-banco.

SQL

```
-- Habilita a geração de UUIDs nativos do Postgres
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Tabela de Usuários (Caso você use JWT/Sessions próprias)
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    full_name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Tabela de Instituições Bancárias
CREATE TABLE institutions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name TEXT NOT NULL,
    bank_code TEXT, -- ex: '077' para Inter, '260' para Nubank
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Tabela de Contas/Cartões
CREATE TABLE accounts (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    institution_id UUID REFERENCES institutions(id) ON DELETE SET NULL,
    name TEXT NOT NULL, -- ex: 'Conta Gastos', 'Cartão Black'
```

```
    type VARCHAR(50) CHECK (type IN ('CONTA_CORRENTE', 'CARTAO_DE_CREDITO',
'INVESTIMENTO')),
    currency VARCHAR(10) DEFAULT 'BRL',
    current_balance DECIMAL(15, 2) DEFAULT 0.00,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

---

2. Transações e Motor de Deduplicação

O campo `transaction_hash` é a chave para evitar dados duplicados entre diferentes arquivos e bancos.

SQL

```
-- Tabela de Categorias
CREATE TABLE categories (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name TEXT UNIQUE NOT NULL,
    color VARCHAR(20) DEFAULT '#808080',
    icon VARCHAR(50),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Tabela de Transações
CREATE TABLE transactions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    account_id UUID NOT NULL REFERENCES accounts(id) ON DELETE CASCADE,

    -- Controle de Duplicidade
    fitid TEXT, -- ID original do banco (OFX)
    transaction_hash TEXT UNIQUE NOT NULL, -- UNIQUE para impedir duplicados

    -- Dados da Transação
    description_raw TEXT NOT NULL,
    description_clean TEXT,
    amount DECIMAL(15, 2) NOT NULL,
    date DATE NOT NULL,

    -- Categorização e IA
    category_id UUID REFERENCES categories(id) ON DELETE SET NULL,
    category_confidence DECIMAL(3, 2),
    needs_review BOOLEAN DEFAULT TRUE,

    -- Conciliação (Transferências entre contas próprias)
    transfer_id UUID REFERENCES transactions(id) ON DELETE SET NULL,

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Índice para busca de conciliação (Valor e Data)
CREATE INDEX idx_trans_recon_search ON transactions (amount, date, user_id);
```

---

3. Cache de Inteligência Artificial

Evita chamadas repetitivas à API de IA, economizando custo e tempo.

SQL

```
CREATE TABLE ai_category_cache (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    description_clean TEXT UNIQUE NOT NULL, -- Ex: 'IFOOD'
    category_id UUID REFERENCES categories(id) ON DELETE CASCADE,
    confidence_score DECIMAL(3, 2),
    occurrence_count INTEGER DEFAULT 1,
    is_global BOOLEAN DEFAULT FALSE,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

---

4. Investimentos e Histórico de Patrimônio

Focado no cálculo do **Net Worth** (Ativos - Passivos).

SQL

```
-- Carteira de Ativos
CREATE TABLE investments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    ticker VARCHAR(20) NOT NULL,
    type VARCHAR(50) NOT NULL, -- ex: 'ACAO', 'FII', 'CDB'
    quantity DECIMAL(18, 8) DEFAULT 0,
    average_price DECIMAL(18, 2) DEFAULT 0,
    current_price DECIMAL(18, 2) DEFAULT 0,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Registro de Patrimônio Mensal
CREATE TABLE net_worth_history (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    snapshot_date DATE NOT NULL,
    total_assets DECIMAL(15, 2) NOT NULL,
    total_liabilities DECIMAL(15, 2) NOT NULL,
    net_worth DECIMAL(15, 2) NOT NULL,
    UNIQUE(user_id, snapshot_date)
);
```

📝 Notas de Implementação para o Desenvolvedor

Como gerar o `transaction_hash` no Node.js:

Para garantir que o hash seja único e consistente, concatene os campos fixos da transação:

JavaScript

```
const crypto = require('crypto');

function generateHash(accountId, fitid, date, amount) {
    const data = `${accountId}|${fitid}|${date}|${amount.toFixed(2)}`;
    return crypto.createHash('sha256').update(data).digest('hex');
}
```

Lógica de Insert (Prevenção de Erros):

Como o `transaction_hash` é `UNIQUE`, se você tentar importar o mesmo arquivo duas vezes, o Postgres retornará um erro. Use `ON CONFLICT` para ignorar:

SQL

```
INSERT INTO transactions (user_id, account_id, transaction_hash, ...)
VALUES (...)
ON CONFLICT (transaction_hash) DO NOTHING;
```

Busca no Cache de IA:

Antes de enviar para o Gemini/Groq, faça:

SQL

```
SELECT category_id, confidence_score
FROM ai_category_cache
WHERE description_clean = 'DESC_DA_TRANSACAO_LIMPA';
```