Documentação da API REST - Financie

Pedro Henrique Silva Quixabeira

10 de Maio de 2025

Contents

1 Introdução

A API REST do Financie é o núcleo da plataforma SaaS, permitindo a interação entre o frontend (React.js) e o backend (Flask com PostgreSQL). Ela suporta as funcionalidades do MVP, incluindo autenticação de usuários, gerenciamento de transações financeiras e visualização de dados no dashboard. A API segue o padrão **OpenAPI 3.0**, é versionada (v1) e utiliza **JSON** para troca de dados. Esta documentação detalha a estrutura, arquitetura, endpoints, payloads, respostas, autenticação, erros e diretrizes de segurança, garantindo conformidade com a LGPD e usabilidade para desenvolvedores.

2 Informações Gerais

2.1 Base URL

- **URL base**: https://api.financie.com.br/api/v1
- **Ambiente de teste**: https://staging.api.financie.com.br/api/v1 (disponível para beta testers)

2.2 Autenticação

- **Método**: Tokens JWT (JSON Web Tokens) gerados pelo Firebase Authentication.
- **Header**: Todas as requisições autenticadas devem incluir:

```
Authorization: Bearer <token>
```

- **Validade do token**: 24 horas. Após expiração, o usuário deve realizar login novamente.
- **Integração com Firebase**: O backend valida tokens usando o SDK Firebase Admin (Python).

2.3 Formato de Dados

- **Requisições e respostas**: JSON.
- **Codificação**: UTF-8.
- **Datas**: Formato ISO 8601 (ex.: `2025-08-01T14:30:00Z`).
- **Valores monetários**: Decimal com duas casas (ex.: `1500.00`).

2.4 Códigos de Status HTTP

- `200 OK`: Requisição bem-sucedida.
- `201 Created`: Recurso criado com sucesso.
- `400 Bad Request`: Dados inválidos ou malformados.
- `401 Unauthorized`: Token ausente ou inválido.
- `403 Forbidden`: Usuário sem permissão.
- `404 Not Found`: Recurso não encontrado.
- `429 Too Many Requests`: Limite de requisições excedido.
- `500 Internal Server Error`: Erro interno do servidor.

2.5 Rate Limiting

- **Limite**: 100 requisições por minuto por IP.
- **Resposta em caso de exceder**: `429 Too Many Requests` com header `Retry-After`.

2.6 Conformidade com LGPD

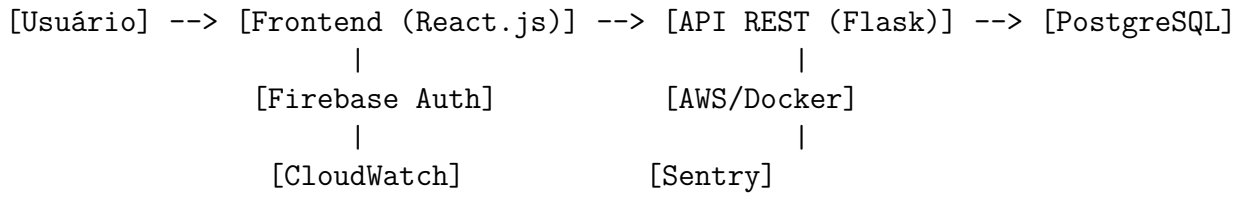
- **Consentimento**: Usuários devem aceitar a política de privacidade no cadastro.
- **Criptografia**: Dados sensíveis (ex.: CPF/CNPJ) criptografados com AES-256 em repouso e TLS 1.2/1.3 em trânsito.
- **Exclusão de dados**: Suporte à exclusão de contas via endpoint específico.
- **Logs**: Acessos auditados por 6 meses, armazenados em conformidade com a LGPD.

3 Estrutura e Arquitetura da API

3.1 Visão Geral

A API REST do Financie é construída como uma aplicação **monolítica modular**, utilizando o framework Flask (Python 3.11) para o backend, PostgreSQL como banco de dados relacional e Firebase Authentication para gerenciamento de autenticação. A arquitetura segue o padrão **MVC (Model-View-Controller)** adaptado para APIs REST, com camadas claras para separação de responsabilidades. A API é projetada para ser escalável, segura e de fácil manutenção, atendendo aos requisitos não funcionais de disponibilidade (99,9%), tempo de resposta (< 2 segundos) e suporte a 100 usuários simultâneos no MVP.

Diagrama de Arquitetura:



(Nota: O diagrama pode ser criado no Figma ou Draw.io e incluído na documentação final.)

3.2 Camadas da Arquitetura

A API é organizada em quatro camadas principais:

1. **Camada de Apresentação (API Endpoints)**:
    - Responsável por receber requisições HTTP, validar entradas e retornar respostas JSON.
    - Implementada com Flask-RESTful, que define recursos (ex.: `/auth`, `/transactions`, `/dashboard`).
    - Utiliza middlewares para autenticação (validação de JWT), rate limiting (Flask-Limiter) e logging.
    - Exemplo: O endpoint `POST /transactions` valida o payload e delega a lógica para a camada de serviço.
2. **Camada de Serviço (Business Logic)**:
    - Contém a lógica de negócio, como validação de transações, cálculos de fluxo de caixa e regras de conformidade com LGPD.
- Organizada em módulos Python (ex.: `services/auth`$_service.py$, `services/transaction`$_servi$
3. **Camada de Dados (Data Access)**:
    - Gerencia a interação com o PostgreSQL usando SQLAlchemy como ORM.
    - Define modelos (ex.: `User`, `Transaction`) mapeados para tabelas do banco.
    - Implementa operações CRUD e consultas agregadas (ex.: soma de receitas por mês).
    - Usa `pgcrypto` para criptografia de dados sensíveis (ex.: CPF/CNPJ).
    - Exemplo: A tabela `transactions` é consultada com índices para otimizar o dashboard.
4. **Camada de Infraestrutura**:
    - Inclui configurações de deploy (AWS Elastic Beanstalk, Docker), monitoramento (Sentry, CloudWatch) e autenticação externa (Firebase).
    - Gerencia conexões com o banco (via SQLAlchemy) e serviços externos (via SDKs).
    - Exemplo: Logs de requisições são enviados ao CloudWatch para auditoria LGPD.

3.3 Estrutura de Código

A estrutura de diretórios do backend reflete a modularidade e separação de responsabilidades:

```
financie-api/
  src/
      controllers/          # Lógica dos endpoints (Flask-RESTful)
          auth_controller.py
          transaction_controller.py
          dashboard_controller.py
      services/             # Lógica de negócio
          auth_service.py
          transaction_service.py
          dashboard_service.py
      models/               # Modelos SQLAlchemy
          user.py
          transaction.py
      middleware/           # Middlewares (autenticação, rate limiting)
          auth_middleware.py
          rate_limit_middleware.py
      utils/                # Funções auxiliares (ex.: validação de CPF)
          validators.py
          crypto.py
      config/               # Configurações (ex.: chaves Firebase, DB)
          settings.py
      main.py               # Ponto de entrada da aplicação
  tests/                    # Testes unitários (Pytest, pós-MVP)
  docs/                     # Documentação Swagger (OpenAPI)
  Dockerfile                # Configuração Docker
  requirements.txt          # Dependências Python
  .env                      # Variáveis de ambiente
```

- **Controllers**: Mapeiam endpoints para funções que processam requisições e retornam respostas.
- **Services**: Encapsulam regras de negócio, chamando modelos ou serviços externos (ex.: Firebase).
- **Models**: Definem a estrutura das tabelas e métodos de acesso a dados.
- **Middleware**: Aplicam validações globais (ex.: verificar JWT em rotas protegidas).
- **Utils**: Funções reutilizáveis, como validação de CPF/CNPJ ou criptografia.

3.4 Padrões de Projeto

- **RESTful Design**: Endpoints seguem convenções REST (ex.: `GET /resources` para listar, `POST /resources` para criar).

- **Repository Pattern**: A camada de dados usa repositórios (ex.: `TransactionRepository`) para abstrair o acesso ao banco, facilitando testes e futuras mudanças (ex.: migração para MongoDB).
- **Dependency Injection**: Serviços recebem dependências (ex.: conexão com banco) via construtores, permitindo mock em testes.
- **Error Handling**: Exceções personalizadas (ex.: `ValidationError`, `NotFoundError`) são capturadas e mapeadas para respostas HTTP consistentes.

3.5 Fluxo de Dados

Exemplo de fluxo para uma requisição `POST /transactions`:

1. O frontend envia uma requisição com JSON e token JWT.
2. O middleware $\texttt{auth}_{m}iddlewarevalidaotokencomoFirebase.O\texttt{transaction}_{c}ontrollerrecebeopayl$
3. A resposta JSON com a transação criada é retornada (status 201).

Diagrama de Sequência:

```
Usuário -> Frontend -> API (Controller) -> Middleware (Auth) -> Firebase
           Frontend <- API <- Service -> Model -> PostgreSQL
```

(Sugestão: Criar no Figma.)

3.6 Integrações

- **Firebase Authentication**:
  - SDK Python valida tokens JWT e gerencia cadastro/login.
- Integração via $\texttt{firebase}_{a}dminno\texttt{auth}_{s}ervice.$
- **PostgreSQL**:
  - Conexão via SQLAlchemy com pool de conexões para 100 usuários simultâneos.
- Índices em $\texttt{user}_{i}de\texttt{date}paraconsultasrpidasnodashboard.$
- **AWS**:
  - Elastic Beanstalk hospeda a API com Gunicorn (4 workers).
  - CloudWatch armazena logs de requisições para auditoria LGPD.
  - RDS gerencia o PostgreSQL com backups diários.
- **Sentry**:
  - Captura erros em tempo real (ex.: falhas em validação ou banco).

3.7 Escalabilidade e Performance

- **Cache**: Consultas frequentes do dashboard serão armazenadas em Redis (pós-MVP).
- **Balanceamento de Carga**: AWS Elastic Beanstalk distribui requisições entre instâncias.

- **Índices**: Otimizam consultas no PostgreSQL (ex.: `SELECT SUM(amount) FROM transactions WHERE user`$_id = ?$).**`Rate Limiting`** : $Flask-Limiter evita sobrecarga com 100 requisies/minuto por IP.$

3.8 Segurança

- **Criptografia**: Colunas sensíveis (ex.: `cpf`$_cnpj$)$criptografadas com$`pgcrypto`.**`Validação`** : $Sanitizao de entradas com Flask-WTF para prevenir SQL Injection e XSS.$
- **LGPD**: Consentimento explícito no cadastro e exclusão de dados via endpoint `/auth/account`.

4 Endpoints

4.1 Autenticação

Gerenciada pelo Firebase Authentication, com endpoints de suporte no backend.

4.1.1 Registro de Usuário

- **Endpoint**: `POST /auth/register`
- **Descrição**: Cria uma nova conta de usuário.
- **Autenticação**: Não requerida.
- **Corpo da Requisição**:

```
{
  "email": "string",
  "password": "string",
  "name": "string",
  "cpf_cnpj": "string"
}
```

  - `email`: E-mail válido (ex.: `joao@exemplo.com`).
  - `password`: Mínimo 8 caracteres, com letras e números.
  - `name`: Nome completo (máximo 100 caracteres).

- `cpf`$_cnpj : CPF(11 dgitos) ou CNPJ(14 dgitos), sem formatao.$
- **Respostas**:

- **201 Created**:

```
{
  "message": "Usuário criado com sucesso. Verifique seu e-
     mail para ativar a conta.",
  "user_id": "uuid"
}
```

- **400 Bad Request**:

```
{
  "error": "E-mail já cadastrado"
}
```

- **400 Bad Request** (validação):

```
{
  "error": "CPF/CNPJ inválido"
}
```

- **Exemplo de Requisição**:

```
curl -X POST https://api.financie.com.br/api/v1/auth/register \
-H "Content-Type: application/json" \
-d '{"email":"joao@exemplo.com","password":"Senha123","name":"
   João Silva","cpf_cnpj":"12345678901"}'
```

4.1.2 Login

- **Endpoint**: `POST /auth/login`
- **Descrição**: Autentica o usuário e retorna um token JWT.
- **Autenticação**: Não requerida.
- **Corpo da Requisição**:

```
{
  "email": "string",
  "password": "string"
}
```

- **Respostas**:
  - **200 OK**:

```
{
  "token": "string",
  "user_id": "uuid",
  "email": "string",
  "name": "string"
}
```

- **401 Unauthorized**:

```
{
  "error": "Credenciais inválidas"
}
```

- **Exemplo de Requisição**:

```
curl -X POST https://api.financie.com.br/api/v1/auth/login \
-H "Content-Type: application/json" \
-d '{"email":"joao@exemplo.com","password":"Senha123"}'
```

4.1.3 Recuperação de Senha

- **Endpoint**: `POST /auth/reset-password`
- **Descrição**: Envia um e-mail com link para redefinição de senha.
- **Autenticação**: Não requerida.
- **Corpo da Requisição**:

```
{
  "email": "string"
}
```

- **Respostas**:
  - **200 OK**:

```
{
  "message": "E-mail de recuperação enviado"
}
```

  - **404 Not Found**:

```
{
  "error": "E-mail não encontrado"
}
```

- **Exemplo de Requisição**:

```
curl -X POST https://api.financie.com.br/api/v1/auth/reset-
   password \
-H "Content-Type: application/json" \
-d '{"email":"joao@exemplo.com"}'
```

4.1.4 Exclusão de Conta (LGPD)

- **Endpoint**: `DELETE /auth/account`
- **Descrição**: Exclui a conta do usuário (soft delete, com retenção de 30 dias).
- **Autenticação**: Requerida (JWT).
- **Corpo da Requisição**: Nenhum.
- **Respostas**:
  - **200 OK**:

```
{
  "message": "Conta marcada para exclusão. Dados serão
     removidos em 30 dias."
}
```

  - **401 Unauthorized**:

```
{
  "error": "Token inválido"
}
```

- **Exemplo de Requisição**:

```
curl -X DELETE https://api.financie.com.br/api/v1/auth/account \
-H "Authorization: Bearer <token>"
```

4.2 Transações

Endpoints para gerenciamento de transações financeiras (receitas e despesas).

4.2.1 Criar Transação

- **Endpoint**: `POST /transactions`
- **Descrição**: Registra uma nova transação (receita ou despesa).
- **Autenticação**: Requerida (JWT).
- **Corpo da Requisição**:

```
{
  "type": "string",
  "amount": "number",
  "date": "string",
  "description": "string",
  "category": "string"
}
```

  - `type`: `receita` ou `despesa`.

- `amount`: Valor positivo (ex.: `1500.00`).
- `date`: Data no formato `YYYY-MM-DD`.
- `description`: Descrição (máximo 200 caracteres).
- `category`: Categoria predefinida (ex.: `Vendas`, `Aluguel`, `Marketing`).

- **Respostas**:
  - **201 Created**:

```
{
  "id": "uuid",
  "type": "receita",
  "amount": 1500.00,
  "date": "2025-08-01",
  "description": "Venda de produto",
  "category": "Vendas",
  "created_at": "2025-08-01T14:30:00Z"
}
```

  - **400 Bad Request**:

```
{
  "error": "Valor deve ser maior que zero"
}
```

- **Exemplo de Requisição**:

```
curl -X POST https://api.financie.com.br/api/v1/transactions \
-H "Authorization: Bearer <token>" \
-H "Content-Type: application/json" \
-d '{"type":"receita","amount":1500.00,"date":"2025-08-01","
   description":"Venda de produto","category":"Vendas"}'
```

4.2.2 Listar Transações

- **Endpoint**: `GET /transactions`
- **Descrição**: Retorna uma lista de transações com filtros opcionais.
- **Autenticação**: Requerida (JWT).
- **Parâmetros de Query**:
  - `start`$_date(opcional)$: $Data inicial$(`YYYY-MM-DD`).`end`$_date(opcional)$: $Data final$(`YYYY-MM-DD`
  - `type` (opcional): `receita` ou `despesa`.
  - `page` (opcional): Página (padrão: 1).
  - `limit` (opcional): Itens por página (padrão: 20, máx.: 100).
- **Respostas**:

- **200 OK**:

```
{
  "transactions": [
    {
      "id": "uuid",
      "type": "receita",
      "amount": 1500.00,
      "date": "2025-08-01",
      "description": "Venda de produto",
      "category": "Vendas",
      "created_at": "2025-08-01T14:30:00Z"
    }
  ],
  "total": 1,
  "page": 1,
  "limit": 20
}
```

- **400 Bad Request**:

```
{
  "error": "Data inválida"
}
```

- **Exemplo de Requisição**:

```
curl -X GET https://api.financie.com.br/api/v1/transactions?
   start_date=2025-08-01&end_date=2025-08-31&category=Vendas \
-H "Authorization: Bearer <token>"
```

4.2.3 Editar Transação

- **Endpoint**: `PUT /transactions/:id`
- **Descrição**: Atualiza uma transação existente.
- **Autenticação**: Requerida (JWT).
- **Parâmetros de URL**:
  - `id`: UUID da transação.
- **Corpo da Requisição**:

```
{
  "type": "string",
  "amount": "number",
  "date": "string",
  "description": "string",
  "category": "string"
}
```

- **Respostas**:
  - **200 OK**:

```
{
  "id": "uuid",
  "type": "receita",
  "amount": 2000.00,
  "date": "2025-08-01",
  "description": "Venda ajustada",
  "category": "Vendas",
  "updated_at": "2025-08-02T10:00:00Z"
}
```

  - **404 Not Found**:

```
{
  "error": "Transação não encontrada"
}
```

- **Exemplo de Requisição**:

```
curl -X PUT https://api.financie.com.br/api/v1/transactions/123
   e4567-e89b-12d3-a456-426614174000 \
-H "Authorization: Bearer <token>" \
-H "Content-Type: application/json" \
-d '{"type":"receita","amount":2000.00,"date":"2025-08-01","
   description":"Venda ajustada","category":"Vendas"}'
```

4.2.4 Excluir Transação

- **Endpoint**: `DELETE /transactions/:id`
- **Descrição**: Remove uma transação.
- **Autenticação**: Requerida (JWT).
- **Parâmetros de URL**:
  - `id`: UUID da transação.
- **Respostas**:
  - **200 OK**:

```
{
  "message": "Transação excluída com sucesso"
}
```

  - **404 Not Found**:

```
{
  "error": "Transação não encontrada"
}
```

- **Exemplo de Requisição**:

```
curl -X DELETE https://api.financie.com.br/api/v1/transactions
   /123e4567-e89b-12d3-a456-426614174000 \
-H "Authorization: Bearer <token>"
```

4.3 Dashboard

Endpoints para dados agregados exibidos no dashboard.

4.3.1 Resumo Financeiro

- **Endpoint**: `GET /dashboard`
- **Descrição**: Retorna dados agregados para o dashboard (fluxo de caixa, resumo por categoria).
- **Autenticação**: Requerida (JWT).
- **Parâmetros de Query**:

– `start`$_{d}ate(opcional)$: $Datainicial$(`YYYY-MM-DD`).`end`$_{d}ate(opcional)$: $Data final$(`YYYY-MM-DD`).

- **Respostas**:

  – **200 OK**:

```
{
  "cash_flow": [
    {
      "month": "2025-08",
      "revenue": 5000.00,
      "expense": 3000.00,
      "balance": 2000.00
    }
  ],
  "categories": [
    {
      "category": "Vendas",
      "amount": 5000.00,
      "type": "receita"
    },
    {
      "category": "Aluguel",
      "amount": 2000.00,
      "type": "despesa"
    }
  ],
  "recent_transactions": [
    {
      "id": "uuid",
      "type": "receita",
```

```
      "amount": 1500.00,
      "date": "2025-08-01",
      "description": "Venda de produto",
      "category": "Vendas"
    }
  ]
}
```

- **400 Bad Request**:

```
{
  "error": "Período inválido"
}
```

- **Exemplo de Requisição**:

```
curl -X GET https://api.financie.com.br/api/v1/dashboard?
   start_date=2025-08-01&end_date=2025-08-31 \
-H "Authorization: Bearer <token>"
```

5 Modelos de Dados

5.1 Usuário

```
{
  "id": "uuid",
  "email": "string",
  "name": "string",
  "cpf_cnpj": "string",
  "created_at": "string (ISO 8601)"
}
```

5.2 Transação

```
{
  "id": "uuid",
  "user_id": "uuid",
  "type": "string (receita | despesa)",
  "amount": "number",
  "date": "string (YYYY-MM-DD)",
  "description": "string",
  "category": "string",
  "created_at": "string (ISO 8601)",
  "updated_at": "string (ISO 8601)"
}
```

6 Erros Comuns

- **Erro de Validação**:

```
{
  "error": "Campo obrigatório ausente",
  "details": {
    "field": "email",
    "message": "E-mail é obrigatório"
  }
}
```

- **Erro de Autenticação**:

```
{
  "error": "Token expirado"
}
```

- **Erro de Servidor**:

```
{
  "error": "Erro interno. Tente novamente mais tarde."
}
```

7 Segurança

- **Criptografia**: Dados sensíveis (ex.: `CPF/CNPJ`) armazenados com AES-256 via `pgcrypto` no PostgreSQL. Comunicação via TLS 1.2/1.3.
- **Validação de Entrada**: Proteção contra SQL Injection e XSS usando sanitização no Flask (Flask-WTF) e validação no frontend.
- **Rate Limiting**: Implementado com Flask-Limiter.
- **LGPD**:
  - Consentimento explícito coletado no cadastro.
  - Exclusão de dados sob demanda.
  - Logs de acesso armazenados por 6 meses em AWS CloudWatch.

8 Ferramentas e Integrações

- **Backend**: Flask (Python 3.11), Flask-RESTful, SQLAlchemy, PyJWT.
- **Banco de Dados**: PostgreSQL 15.x (Amazon RDS).
- **Autenticação**: Firebase Authentication (SDK Python).

- **Documentação**: Swagger/OpenAPI, gerado automaticamente com Flask-RESTful.
- **Monitoramento**: Sentry para erros, AWS CloudWatch para logs e métricas.

9 Planejamento para Futuras Versões

- **v1.1 (Q4 2025)**: Suporte a notificações por e-mail para transações (ex.: alerta de despesa alta).
- **v2.0 (2026)**: Integração com Pix (API Banco Central), relatórios exportáveis (PDF/CSV), insights baseados em IA (ex.: integração com Grok da xAI).

10 Como Testar

- **Ambiente de Teste**: Use o ambiente staging (https://staging.api.financie.com.br/api/v1).
- **Ferramentas Recomendadas**:
  - Postman ou Insomnia para chamadas manuais.
  - cURL para scripts automatizados.
- **Chave de Teste**: Solicite uma chave de teste para beta testers via `suporte@financie.com.b`
- **Exemplo de Fluxo**:
  1. Registre um usuário (`POST /auth/register`).
  2. Faça login (`POST /auth/login`) para obter o token.
  3. Crie uma transação (`POST /transactions`).
  4. Visualize o dashboard (`GET /dashboard`).

11 Suporte

- **E-mail**: `suporte@financie.com.br`
- **Documentação Swagger**: Disponível em https://api.financie.com.br/api/v1/docs (a partir de agosto 2025).
- **Comunidade**: Grupo de beta testers no WhatsApp (convite via landing page).