**🛠️ Tecnologias Sugeridas para o Desenvolvimento do Aplicativo de Gestão Financeira Pessoal**

**✅ Visão Geral da Arquitetura**

* **Frontend:** Mobile-first, com possibilidade de também rodar como Web App (PWA).
* **Backend:** API RESTful
* **Banco de Dados:** Relacional, com foco em segurança dos dados.
* **Autenticação:** JWT com OAuth2 para futuras integrações bancárias (Open Finance).

**✅ Frontend (Mobile e Web - React Native com Web Support)**

| **Camada** | **Tecnologias/Bibliotecas** |
| --- | --- |
| Linguagem principal | **JavaScript / TypeScript** |
| Framework Mobile | **React Native** (com Expo para acelerar o desenvolvimento) |
| Web Support | **React Native for Web** ou criar uma versão web com **Next.js** se quiser uma UX mais avançada |
| Component Library | **React Native Paper** ou **Shadcn/UI** (com Tailwind se quiser visual moderno) |
| Gerenciamento de Estado | **Zustand** ou **Redux Toolkit** |
| Navegação | **React Navigation** |
| Gráficos | **Recharts**, **Victory Native**, ou **React Native Chart Kit** |
| Formulários | **React Hook Form** ou **Formik** |
| Validação | **Yup** |
| Autenticação | Firebase Auth (ou Auth0) como primeiro passo |
| Push Notifications | **Expo Notifications** (ou Firebase Cloud Messaging depois) |

**✅ Backend (API RESTful**

| **Camada** | **Tecnologias** |
| --- | --- |
| Linguagem | **Python (Django REST Framework)** |
| Framework | Se Python: **Django REST Framework** |
| Banco de Dados | **PostgreSQL** (recomendado pela robustez com dados financeiros) |
| ORM | **Django ORM (se Python)** |
| Autenticação | **JWT (JSON Web Tokens)** |
| Integração Bancária | **Open Finance APIs** ou serviços como **Belvo**, **Plaid** ou **Banco do Brasil Open Banking API** (se quiser integração futura) |
| Upload de Arquivos (ex: CSV de extratos) | **Django File Upload** |
| API de Notificações | **Firebase Cloud Messaging** |
| Criação de Gráficos Server-side (se precisar de PDFs, por exemplo) | **Matplotlib (se Python)** |

**✅ Banco de Dados**

| **Opção** | **Detalhes** |
| --- | --- |
| **PostgreSQL** | Alta segurança, tipos nativos para data/valores numéricos, escalável |
|  |  |

**✅ Infraestrutura / Deployment**

| **Camada** | **Tecnologias** |
| --- | --- |
| Hospedagem Backend | **Heroku**, **Render**, **AWS Elastic Beanstalk** |
| Banco de Dados Cloud | **Supabase**, **ElephantSQL**, ou **AWS RDS** |
| Armazenamento de Arquivos | **AWS S3** ou **Firebase Storage** |
| Autenticação | **Firebase Auth**, **Auth0** ou implementar próprio via JWT |
| CI/CD | **GitHub Actions**, **Bitrise** ou **Expo EAS Build** (se usar React Native Expo) |

**✅ Bibliotecas Adicionais (UX, Segurança, Performance)**

| **Necessidade** | **Bibliotecas** |
| --- | --- |
| Segurança de Senhas | **bcryptjs** ou **argon2** |
| Validação de Dados Backend | **Joi**, **class-validator** (NestJS) ou **Django REST Validators** |
| Monitoramento de Erros | **Sentry** |
| Cache API | **Redis** (se necessário performance extra) |
| Rate Limiting (segurança de API) | **express-rate-limit** (Node) ou **django-ratelimit** |
| Documentação da API | **Swagger (OpenAPI)** com **NestJS** ou **drf-yasg (Django)** |

**✅ Resumo Final do Stack Sugerido**

| **Área** | **Ferramentas** |
| --- | --- |
| Frontend Mobile/Web | React Native (Expo), React Native for Web ou Next.js |
| Backend | Python (Django REST) |
| Banco de Dados | PostgreSQL |
| Autenticação | JWT + Firebase/Auth0 |
| Gráficos | Recharts / Victory / Chart.js |
| Notificações | Firebase |
| Deployment | Render / AWS / Vercel / Supabase |