Sistema de Gastos Personales — Especificacion Tecnica (v1.0)

Fecha: 22 de octubre de 2025

# Resumen y Objetivos

Aplicacion full-stack para registrar ingresos y gastos, visualizar balances y generar reportes. Este documento define arquitectura de frontend y backend, despliegue en nube, flujo de Git/GitHub, seccion de perfil de usuario y registro con email o numero de telefono.

# Alcance del Proyecto

## MVP

* + Registro, login y logout con email/contraseña.
  + CRUD de transacciones (ingresos/gastos) con categorias.
  + Balance por periodo (dia/semana/mes) y acumulado.
  + Reportes con Chart.js (barra, dona, linea).
  + API REST documentada con Swagger/OpenAPI.
  + Persistencia en SQLite (dev) y Postgres (prod).

## PLUS

* + Registro con telefono (OTP) y 2FA (TOTP).
  + Presupuestos por categoria y alertas de sobre-gasto.
  + Cuentas multiples y transferencias internas.
  + Transacciones recurrentes y adjuntos (tickets) en object storage.
  + Multi-moneda, i18n y PWA (offline-first).

# Arquitectura — Vision General

**Cliente**: SPA en JavaScript (Vite + React) con Chart.js.

**Servidor**: API RESTful en .NET 8 (Clean Architecture) o Java Spring Boot 3 (Hexagonal).

**BD**: SQLite (dev) y PostgreSQL (prod).

**Autenticacion**: JWT (access/refresh). Opcional: Firebase Auth para login con telefono.

**Infra**: Docker y docker-compose para desarrollo; despliegue en nube gratuita.

# Arquitectura de Frontend (JavaScript)

## Stack sugerido

* + Vite + React 18 (alternativa: Vanilla JS).
  + CSS: Tailwind (opcional) o CSS Modules.
  + Chart.js para visualizaciones.
  + React Router para ruteo (login, dashboard, transacciones, perfil).
  + Manejo de estado: Context + Reducers (o Zustand).
  + Cliente HTTP: fetch/axios con interceptores para JWT (refresh).
  + Validacion de formularios con reglas HTML5 y utilidades personalizadas.

***Estructura de carpetas***

frontend/ src/ app/ (providers, rutas) features/ auth/ (pantallas y hooks) transactions/ (CRUD, filtros) reports/ (graficas) profile/ (mi perfil) shared/ (ui, componentes comunes, utilidades) public/ .env

# Arquitectura de Backend — .NET 8 (Clean Architecture)

## Capas

* + Domain: entidades (User, Account, Category, Transaction, Budget), value objects, reglas.
  + Application: DTOs, casos de uso (Handlers/Services), interfaces de repositorio, validaciones.
  + Infrastructure: EF Core, repositorios, acceso a BD (Postgres/SQLite), servicios externos (email, storage).
  + Presentation: Controllers, mapeos (AutoMapper), filtros, Swagger.

***Estructura***

backend-dotnet/ src/ Domain/ Application/ Infrastructure/ Presentation/ (WebApi) tests/ docker/ appsettings.json y appsettings.Development.json

***Endpoints clave (REST)***

|  |  |  |
| --- | --- | --- |
| Metodo | Ruta | Descripcion |
| POST | /api/auth/register | Registro (email/password) |
| POST | /api/auth/login | Login y emision de tokens |
| POST | /api/auth/refresh | Refrescar token |
| POST | /api/auth/phone/start | Iniciar OTP a telefono (si se usa Firebase) |
| POST | /api/auth/phone/verify | Verificar OTP y autenticar |
| GET | /api/users/me | Obtener perfil |
| PATCH | /api/users/me | Actualizar perfil (nombre, moneda, idioma, avatar) |
| GET | /api/categories | Listar categorias |
| POST | /api/categories | Crear categoria |
| GET | /api/transactions | Listar/filtrar transacciones |
| POST | /api/transactions | Crear transaccion |
| GET | /api/reports/summary | Resumen (por periodo y categoria) |

# Alternativa Backend — Java Spring Boot 3 (Hexagonal)

Capas: domain (entidades/puertos), application (servicios/casos de uso), adapters (web, persistence). Spring Security con JWT, Spring Data JPA (Hibernate) y PostgreSQL.

# Autenticacion y Registro (Email / Telefono)

## Email/Password (MVP)

* + Registro: email unico, contraseña con politica (minimo 8, complejidad).
  + Verificacion por enlace de correo (opcional).
  + Login: JWT access (15 min) y refresh (7 dias).

## Telefono (OTP) — Recomendado: Firebase Auth

Frontend usa Firebase para enviar OTP SMS (reCAPTCHA) y obtener un ID Token; el backend verifica ese token y crea/inicia sesion del usuario. Nota: los SMS pueden tener costo; si se busca 100% gratuito, usar email OTP o TOTP 2FA.

# Modulo Mi Perfil (Requisitos)

* + Campos: nombre, email, telefono, moneda preferida, idioma (es/en), tema (claro/oscuro).
  + Cambiar contraseña y (opcional) activar 2FA (TOTP).
  + Subir avatar (storage de objetos).
  + Verificacion de email y telefono.
  + Eliminar cuenta.

# Despliegue en Nube (gratuita/hobby)

## Opcion A (sugerida)

Frontend: Vercel (plan Hobby) — despliegue automatico desde GitHub. Backend: Fly.io o Railway — servicio web Dockerizado. Base de datos: Neon.tech (PostgreSQL free tier). Storage de adjuntos (opcional): Cloudflare R2 o S3 (free tier).

## Opcion B

Frontend: Netlify. Backend: Render (servicio web Docker) — sujeto a disponibilidad del plan gratuito. BD: Supabase (Postgres + Auth), alternativa todo-en-uno.

# Variables de entorno (ejemplo)

BACKEND: ASPNETCORE\_ENVIRONMENT=Production

ConnectionStrings Default=Host=...;Database=...;Username=...;Password=... Jwt Issuer=expenses.api Jwt Audience=expenses.web Jwt Key=SECRET\_LARGO Firebase ProjectId=... Firebase CredentialsJson=...(si aplica) FRONTEND: VITE\_API\_BASE\_URL=https://api.tuapp.com VITE\_FIREBASE\_API\_KEY=...(si aplica)

1. **Git y GitHub — Repositorio y Flujo**

## Estructura del repo (monorepo)

repo/ frontend/ backend-dotnet/ (o backend-java/) infra/ (docker-compose, IaC opcional)

.github/workflows/ (CI/CD) README.md, LICENSE, CODE\_OF\_CONDUCT.md, CONTRIBUTING.md

.gitignore

***Estrategia de ramas***

* + Trunk-based: rama main protegida; ramas feature/ para cambios.
  + PRs con revisiones, checks de CI y cobertura minima (por ejemplo, 70%).
  + Conventional Commits y changelog semantico.

## CI/CD (GitHub Actions)

Jobs: lint, test y build en cada PR. Deploy a Vercel (frontend) y Fly/Railway (backend) al hacer push a main. Publicar imagenes Docker en GitHub Container Registry.