App Finanzas Personales — Diseño técnico

Stack objetivo: Supabase (Postgres + Auth + Realtime + Storage) + Flutter (MVVM) + provider + go_router

---

1. Resumen

Diseño para una app móvil PFM (iOS/Android) que permite registrar ingresos/gastos, gestionar categorías y presupuestos, visualizar gráficas mensuales y recibir alertas. Entregables técnicos: esquema de DB (Supabase), historias de usuario, lógica de negocio, arquitectura MVVM (models/viewmodels/views), ejemplos de código para Flutter (provider + go_router) y recomendaciones para features "nice-to-have".

---

2. Entidades (Tablas) y atributos (Postgres)

> Nota: todos los `id` son `uuid` generados por Postgres (`gen_random_uuid()`), `created_at` y `updated_at` timestamps UTC.

users

- id uuid PRIMARY KEY
- email text UNIQUE NOT NULL
- user_name text
- full_name text
- locale text
- currency text (ej. "VES", "USD")
- created_at timestamptz DEFAULT now()
- updated_at timestamptz

> En Supabase usarás la tabla de `auth.users` y esta tabla para perfil extendido si lo prefieres.

accounts (cuentas/monederos)

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id) ON DELETE CASCADE
- name text NOT NULL
- type text CHECK (type IN ('cash','bank','card','wallet','other'))
- currency text
- balance numeric(14,2) DEFAULT 0
- is_default boolean DEFAULT false
- created_at timestamptz DEFAULT now()

categories

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id) ON DELETE CASCADE
- name text NOT NULL
- type text CHECK(type IN ('expense','income','transfer'))
- color text (hex) NULL
- icon text NULL
- created_at timestamptz DEFAULT now()

transactions

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id) ON DELETE CASCADE
- account_id uuid REFERENCES accounts(id)
- category_id uuid REFERENCES categories(id)
- type text CHECK (type IN ('expense','income','transfer')) NOT NULL
- amount numeric(14,2) NOT NULL CHECK (amount >= 0)
- currency text
- note text
- date date NOT NULL -- fecha de la operación
- recurring_rule text NULL -- e.g. 'monthly' / rrule
- metadata jsonb NULL
- created_at timestamptz DEFAULT now()
- updated_at timestamptz

budgets

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id) ON DELETE CASCADE
- name text
- category_id uuid REFERENCES categories(id) NULL -- presupuesto por categoria o NULL para global
- amount numeric(14,2) NOT NULL
- period text CHECK(period IN ('monthly','weekly','yearly')) DEFAULT 'monthly'
- start_date date
- end_date date NULL
- alert_threshold numeric(5,2) NULL -- porcentaje p.ej 80 -> 80%
- created_at timestamptz DEFAULT now()

notifications

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id) ON DELETE CASCADE
- type text CHECK(type IN ('budget_alert','low_balance','reminder','custom'))
- payload jsonb -- datos para la notificación
- delivered boolean DEFAULT false
- delivered_at timestamptz NULL
- scheduled_at timestamptz NULL

- created_at timestamptz DEFAULT now()

exports

- id uuid PRIMARY KEY
- user_id uuid REFERENCES users(id)
- type text CHECK(type IN ('csv','pdf'))
- file_path text -- ruta en Supabase Storage
- params jsonb -- filtros usados para generar
- created_at timestamptz DEFAULT now()

---

3. Relaciones (ER simple)

- users 1..* accounts
- users 1..* categories
- users 1..* transactions
- accounts 1..* transactions
- categories 0..* transactions
- users 1..* budgets
- budgets optional -> category (uno o global)

---

4. SQL ejemplo (creación básica)

```sql
-- Habilitar extensión uuid/ossp o pgcrypto
create extension if not exists pgcrypto;

create table users (
  id uuid primary key default gen_random_uuid(),
  email text unique not null,
  user_name text,
  full_name text,
  locale text,
  currency text default 'VES',
  created_at timestamptz default now()
);

create table accounts (
  id uuid primary key default gen_random_uuid(),
  user_id uuid references users(id) on delete cascade,
  name text not null,
  type text default 'cash',
  currency text,
  balance numeric(14,2) default 0,
  is_default boolean default false,
```

```
  created_at timestamptz default now()
);

-- transactions, categories, budgets similar (omito por brevedad)
```

---

5. Reglas RLS (sugerencia mínima)

- Habilitar RLS en tablas sensibles (accounts, transactions, categories, budgets, notifications, exports)
- Política: `allow select/insert/update/delete where user_id = auth.uid()` (mapear `auth.uid()` al `users.id` o usar `auth` table de Supabase).

Ejemplo:

```
alter table transactions enable row level security;
create policy "user_is_owner" on transactions
  for all using (user_id = auth.uid());
```

> Nota: si usas perfiles en `public.users` distintos de `auth.users`, necesitarás una función que resuelva `auth.uid()` a tu `users.id`.

---

6. Historias de usuario (epics -> historias)

Epic: Onboarding y autenticación

- Como usuario quiero crear una cuenta con email/contraseña para acceder a mis finanzas.
- Como usuario quiero poder crear un perfil básico (nombre, moneda) al iniciar.

Epic: Registro de transacciones

- Como usuario quiero agregar un gasto con categoría, nota y fecha para llevar control.
- Como usuario quiero editar/eliminar transacciones.
- Como usuario quiero marcar transacción como recurrente (mensual) y que se genere automáticamente.

Epic: Cuentas y saldos

- Como usuario quiero crear múltiples cuentas (efectivo, banco, tarjeta).
- Como usuario quiero ver saldo por cuenta y saldo total consolidado.

Epic: Presupuestos y alertas

- Como usuario quiero establecer un presupuesto mensual por categoría.
- Como usuario recibir una alerta cuando lleve >80% del presupuesto.

Epic: Visualización

- Como usuario quiero ver gráficas mensuales (por categoría y por cuenta).
- Como usuario quiero filtrar transacciones por fecha, categoría y cuenta.

Epic: Exportes

- Como usuario quiero exportar periodo en CSV/PDF.

Epic: Nice-to-have

- Conexión con open-banking para sincronizar movimientos.
- Recomendaciones automáticas de ahorro (IA).

---

7. Lógica de negocio (reglas clave)

- Al insertar `transaction` de tipo `expense` restar `amount` del `account.balance` (y sumar si `income`).
- Si `transfer`, hacer 2 transacciones (origen y destino) y no afectar presupuestos.
- Recurrent transactions: al crear con `recurring_rule`, crear tarea en `scheduler` (p.ej. cron o Supabase scheduled functions) para insertar nuevas transacciones.
- Budget check: desencadenar cálculo cada vez que hay una transacción y, si `spent / budget.amount >= alert_threshold`, crear `notification`.
- Conversión de moneda: almacenar `currency` por transacción; conversión on-demand para reportes (usar tasas externas si es necesario).

---

8. Arquitectura MVVM (Flutter)

Estructura de carpetas sugerida

```
/lib
  /src
    /models
      user.dart
      account.dart
      transaction.dart
      category.dart
      budget.dart
    /services
      supabase_service.dart
      notification_service.dart
      export_service.dart
    /viewmodels
      auth_viewmodel.dart
```

```
      home_viewmodel.dart
      transaction_viewmodel.dart
      budget_viewmodel.dart
    /views
      /auth
        login_view.dart
        signup_view.dart
      /home
        home_view.dart
        accounts_view.dart
      /transactions
        transactions_list_view.dart
        transaction_form_view.dart
    /widgets
    /routes
      app_router.dart
    main.dart
```

Reglas para View / ViewModel

- Views: widgets puros que consumen `ChangeNotifier` provistos por `provider`.
- ViewModels: extienden `ChangeNotifier`, encapsulan estado y lógica (invocan `services`).
- Models: POJO con `fromJson` / `toJson` y validaciones ligeras.

---

9. Ejemplos de código (esqueleto)

Model - transaction.dart

```dart
class TransactionModel {
  final String id;
  final String userId;
  final String accountId;
  final String? categoryId;
  final String type; // 'expense'|'income'|'transfer'
  final double amount;
  final DateTime date;
  final String? note;

  TransactionModel({/* campos */});

  factory TransactionModel.fromJson(Map<String,dynamic> json) => ...;
  Map<String,dynamic> toJson() => ...;
}
```

ViewModel - transaction_viewmodel.dart

```
class TransactionViewModel extends ChangeNotifier {
  final SupabaseService _supabase;
  List<TransactionModel> transactions = [];
  bool loading = false;

  TransactionViewModel(this._supabase);

  Future<void> loadTransactions({DateTime? from, DateTime? to}) async {
    loading = true; notifyListeners();
    transactions = await _supabase.fetchTransactions(from: from, to: to);
    loading = false; notifyListeners();
  }

  Future<void> addTransaction(TransactionModel t) async {
    await _supabase.insertTransaction(t);
    // actualizar balances, budgets -> supabase functions o lógica local
    await loadTransactions();
  }
}
```

SupabaseService (sintético)

```
class SupabaseService {
  final SupabaseClient client;
  SupabaseService(this.client);

  Future<List<TransactionModel>> fetchTransactions({DateTime? from, DateTime?
to}) async {
    final resp = await client
      .from('transactions')
      .select()
      .eq('user_id', client.auth.currentUser!.id)
      .order('date', ascending: false)
      .execute();
    return resp.data.map(...).toList();
  }

  Future<void> insertTransaction(TransactionModel t) async {
    await client.from('transactions').insert(t.toJson()).execute();
    // opcional: call edge function to update balances/notifications
  }
}
```

---

10. Routing (go_router) + provider (esquema)

- `main.dart` inicializa `Supabase` y `Provider` con ViewModels.
- `app_router.dart` define rutas: `/login`, `/signup`, `/home`, `/transactions/new`, `/settings`

Ejemplo minimal:

```
final _router = GoRouter(
  routes: [
    GoRoute(name: 'login', path: '/login', builder: (_,__) => LoginView()),
    GoRoute(name: 'home', path: '/', builder: (_,__) => HomeView()),
  ],
  redirect: (context, state) {
    final logged = Provider.of<AuthViewModel>(context, listen:false).isLogged;
    if (!logged && state.location != '/login') return '/login';
    return null;
  }
);
```

---

11. Notas sobre notificaciones

- Para alertas inmediatas (budget/low balance): crear `notification` row en DB y usar Realtime + push (FCM) para enviar al dispositivo.
- Alternativa: para simples recordatorios, programar notificaciones locales en el cliente.
- Supabase Edge function o serverless job calcula y lanza notificaciones (o usar triggers + background workers).

---

12. Export CSV/PDF

- Generar CSV en backend (Edge Function o SQL) y guardar en Supabase Storage; guardar referencia en `exports`.
- Para PDF usar un servicio (puppeteer o librería server-side) o generar localmente en Flutter y subir archivo.

---

13. Seguridad y cumplimiento

- Asegurar RLS robusto.
- Encriptar datos sensibles si aplica.
- Permisos de Storage: sólo el usuario puede leer sus exports.
- Revisión legal para open-banking según país.

14. Recomendaciones y roadmap de MVP (4 semanas sugeridas)

1. Semana 1: Autenticación, modelos Users/Accounts/Categories/Transactions básicas + UI list/add transaction.
2. Semana 2: Saldos por cuenta, budgets simples, RLS y tests básicos.
3. Semana 3: Gráficas mensuales, alertas (local) y export CSV.
4. Semana 4: Pulir UX, crear prototipo Figma, test con usuarios piloto y recopilar feedback.

---

15. Entregables técnicos que puedo generar si quieres

- Archivos SQL para crear las tablas y políticas RLS completas.
- Código base Flutter (esqueleto) con provider + go_router y los ViewModels principales.
- Ejemplo de Edge function en JS/TS para procesar balances y notificaciones.
- Plantilla de Figma (pantallas clave) o recomendaciones de UX.

Si quieres, genero ahora los **scripts SQL completos** y el **esqueleto del proyecto Flutter** (archivos `main.dart`, `app_router.dart`, ejemplos de `viewmodel` y `service`) listos para copiar en tu proyecto.