🎯Objetivo del MVP

Aplicación móvil (iOS/Android) para usuarios sin control de gastos que permita: - Registrar gastos por **texto** o **voz** con comandos simples. - **Extraer** monto/fecha/nota y **clasificar** el gasto automáticamente. - Mostrar **dashboard mensual** y **recomendaciones** de presupuesto. - Confirmación/corrección de categoría para **mejorar el modelo**.

🧩Alcance funcional (MVP)

1. **Onboarding** (explica el “mes de registro”).
2. **Registro de gasto** por:
3. Texto: “gasté 120 en tacos”, “$350 super”, “Uber 95 ayer”.
4. Voz: grabación → transcripción → mismo flujo de texto.
5. **Categorización automática** (top 8–12 categorías, editable por el usuario).
6. **Dashboard** (mes en curso y anterior):
7. Total gastado, % por categoría, top comercios/notas.
8. Tendencia semanal (línea) y distribución (pastel/barras).
9. **Recomendaciones** simples:
10. “Si reduces Ocio 10%, ahorras $X este mes”.
11. Alertas de sobrepaso de umbral por categoría.
12. **Perfil** con meta de ahorro mensual.

🎞Arquitectura (alto nivel)

Stack sugerido - **Frontend móvil**: React Native + Expo. - **API**: FastAPI (Python 3.11+). - **DB**: PostgreSQL 15. - **ORM**: SQLAlchemy + Alembic (migraciones). - **Autenticación**: JWT (access/refresh) + password hashing (argon2/bcrypt). - **Almacenamiento de audio**: S3-compatible (min.io en dev / AWS S3 en prod). - **Colas**: Redis/Sidekiq-like (RQ/Celery) para jobs de transcripción. - **Monitorización**: Sentry (app + backend) y OpenTelemetry (traces). - **Infra**: Render/Railway para MVP; luego AWS (API Gateway + Lambda/ECS + RDS).

```
flowchart LR
  A[App React Native] -- REST/JSON --> B[FastAPI API Gateway]
  A -- Auth --> B
  B --> C[(PostgreSQL)]
  B -- enqueue job --> D[Worker Celery/RQ]
  A -- upload audio --> E[(S3 Storage)]
  D -- read audio --> E
  D -- Speech-to-text --> F[Whisper API ó Vosk local]
  D -- NLP pipeline --> G[Clasificador categorías]
  G -- write --> C
```

```
  B -- read aggregates --> C
  B -- send --> A
```

🎙️ Secuencia de registro de gasto (voz)

```
sequenceDiagram
  participant User
  participant App as App RN
  participant API as FastAPI
  participant S3 as S3
  participant Worker as Worker STT+NLP
  participant DB as PostgreSQL

  User->>App: Tap "Agregar gasto por voz"
  App->>S3: Sube audio (pre-signed URL)
  App->>API: POST /expenses (audio_url)
  API->>DB: Crea expense (status=pending)
  API->>Worker: Enqueue(job_id, audio_url)
  Worker->>S3: Lee audio
  Worker->>Worker: STT (Whisper/Vosk)
  Worker->>Worker: Extracción monto/fecha/notas
  Worker->>Worker: Clasificación categoría
  Worker->>DB: Actualiza expense (status=processed)
  API-->>App: 202 Accepted + expense_id
  App->>API: GET /expenses/{id}
  API->>DB: Consulta expense
  API-->>App: expense con categoría sugerida
  App->>API: PATCH /expenses/{id} (confirm/cambio categoría)
```

Modelo de datos (PostgreSQL)

```
-- usuarios
CREATE TABLE users (
  id UUID PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- catálogo de categorías
```

```sql
CREATE TABLE categories (
  id SERIAL PRIMARY KEY,
  slug TEXT UNIQUE NOT NULL,
  display_name TEXT NOT NULL,
  budget_pct_hint NUMERIC NULL -- % recomendado opcional
);

-- alias/keywords para clasificación rápida
CREATE TABLE category_alias (
  id SERIAL PRIMARY KEY,
  category_id INT REFERENCES categories(id),
  alias TEXT NOT NULL
);

-- gastos
CREATE TABLE expenses (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  amount_cents INT NOT NULL,
  currency CHAR(3) NOT NULL DEFAULT 'MXN',
  expense_date DATE NOT NULL,
  note TEXT NULL,
  raw_text TEXT NULL,
  audio_url TEXT NULL,
  category_id INT REFERENCES categories(id),
  category_model_score NUMERIC NULL,
  status TEXT NOT NULL CHECK (status IN ('pending','processed')),
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- metas/umbral por categoría (opcional MVP)
CREATE TABLE budgets (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  month DATE NOT NULL,
  category_id INT REFERENCES categories(id),
  limit_cents INT NOT NULL
);

-- recomendaciones generadas
CREATE TABLE suggestions (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  month DATE NOT NULL,
  kind TEXT NOT NULL, -- e.g., 'reduce_category','set_goal'
  payload JSONB NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Categorías iniciales (12): comida_super, comida_fuera, transporte, vivienda, servicios, salud, educación, ocio, compras, suscripciones, transferencias, otros.

IA del MVP

1) Transcripción (STT)

- **Opción A (rápida y precisa)**: Whisper API (cloud). Pros: calidad; Contras: costo por minuto.
- **Opción B (on-device/servidor)**: Vosk + modelo español. Pros: costo bajo; Contras: menor precisión.

2) Extracción de entidades (monto/fecha)

- **Reglas + heurísticas** (rápidas y fiables):
- Monto: regex de MXN (`\$\s?([0-9]{1,3}(?:,[0-9]{3})*(?:\.[0-9]{1,2})?)`).
- Fecha: “hoy”, “ayer”, “antier”, o `dd/mm` , `dd-mm` → resolver con timezone.
- Si falta fecha, usar `today()` .

3) Clasificación de categoría

- **Primera versión**: `sklearn` (LogReg/SVM) con TF-IDF + n-grams en español.
- **Upgrade**: `distilbert-multilingual` fine-tuneado (Hugging Face) para mejorar recall.
- **Aprendizaje activo**: guardar confirmaciones/cambios del usuario → retrain semanal.

4) Generación de recomendaciones

- **Heurísticas** sobre agregados:
- Si categoría X > umbral (p. ej., 30% del gasto), sugerir reducir 10%.
- Detectar “suscripciones silenciosas” por repetición de notas similares.
- Sugerir meta de ahorro = 10–20% de ingresos (ingreso declarado por usuario en onboarding).

Nota: **sin conexión bancaria**, optimizar la **fricción baja** de registro y buena UX de confirmación.

🎄 API (FastAPI) — Endpoints principales

```
POST   /auth/register
POST   /auth/login
POST   /auth/refresh

GET    /categories

POST   /expenses                # crea expense (texto o audio_url)
```

```
GET    /expenses?month=YYYY-MM
GET    /expenses/{id}
PATCH  /expenses/{id}          # confirm/update category, note, date
DELETE /expenses/{id}

GET    /analytics/summary?month=YYYY-MM
GET    /analytics/trends?months=6
GET    /recommendations?month=YYYY-MM

POST   /uploads/presign        # retorna URL prefirmada para S3
```

Contrato de `POST /expenses` **(texto):**

```
{
  "raw_text": "gasté 120 en tacos ayer",
  "timezone": "America/Mexico_City"
}
```

Respuesta:

```
{
  "id": "uuid",
  "status": "processed",
  "amount_cents": 12000,
  "expense_date": "2025-08-14",
  "category": {"id": 2, "slug": "comida_fuera", "score": 0.82},
  "note": "tacos"
}
```

Contrato de `POST /expenses` **(audio):**

```
{
  "audio_url": "https://s3/.../clip_123.m4a",
  "timezone": "America/Mexico_City"
}
```

Respuesta 202: `{ "id": "uuid", "status": "pending" }`

🪷 Seguridad y privacidad

- Hash de contraseñas con **argon2** (o bcrypt) y **JWT** con rotación (access 15 min, refresh 14 días).

- Cifrado en tránsito (HTTPS) y en reposo (S3 server-side, RDS encryption).
- **PII mínima**: email y gastos; no almacenar audios tras procesar (configurable).
- **Rate limiting** por IP/usuario (ej. 60 req/min) y validación de tamaño de audio.
- **Logging** con scrubbing de PII en notas.

🧨 Calidad (testing)

- **Unit tests**: extracción regex, normalización de montos/fechas, clasificadores.
- **Integration tests**: endpoints clave con DB temporal (pytest + httpx + docker-compose).
- **E2E**: flujos básicos en app (Detox/Playwright para RN).

🖌️ Deploy y CI/CD

- **Dev**: Docker Compose (api, db, worker, redis, minio).
- **CI**: GitHub Actions → lint (ruff), tests, build images.
- **Prod**: Render/Railway (mínimo). Escalado posterior a AWS: RDS, ECS Fargate, S3, ElastiCache.
- **Observabilidad**: Sentry + Health checks.

🏑 Diseño de UX (puntos clave)

- **Registro en 2 taps**: monto auto-detectado, teclado numérico, categoría sugerida editable.
- **Confirmación post-gasto**: chip de categoría + score (“Sugerido: Comida fuera 82%”).
- **Feedback divertido**: barras de progreso por categoría; logros por racha de registro.

🏸 Roadmap y entregables

Semana 1–2 - Esqueleto RN + FastAPI. - DB y migraciones. - Endpoint /expenses (texto) con extracción y categorización TF-IDF.

Semana 3–4 - Flujo de audio: presigned URL + worker + STT. - Dashboard mensual básico. - Autenticación JWT.

Semana 5–6 - Recomendaciones heurísticas. - Confirmación/corrección de categoría y registro de feedback. - Telemetría + Sentry.

Semana 7–8 - Pulido UX, pruebas E2E, beta cerrada. - Publicación en TestFlight/Play Store (closed testing).

Métricas de éxito (MVP)

- **DAU/WAU**, ratio de registro de gasto por día activo.
- % de gastos **confirmados sin cambios** (precisión percibida).
- Tasa de retención día 7 y día 30.
- % de usuarios que configuran **meta de ahorro**.

Monetización (experimentos post-MVP)

- **Freemium**: límite de categorías/mes gratis; premium con ilimitado + recomendaciones avanzadas ($59–99 MXN/mes).
- **Bundles**: anual con 20–30% descuento.
- **Trials**: 14 días premium.
- **A/B**: copy de valor (“ahorra 10% mensual en 30 días”).

Especificaciones técnicas adicionales

- **Normalización de monto**: eliminar separadores, parse MXN por defecto, permitir “usd, dólares” → convertir si provees FX (más adelante).
- **Fechas**: resolver deícticos (“ayer”, “antier”) con timezone `America/Mexico_City`.
- **Idioma**: español MX; soporte a sinónimos en alias (tacos, comida, restaurante, fonda, antojitos…).
- **Fallback de categoría**: si score < 0.5 → “otros”.
- **Retentiva de audio**: borrar 24h después de procesado (config).

🧨 Dataset inicial para clasificación

- Crear **semilla de 1,000 frases** sintéticas por categoría con variaciones de monto, sinónimos y ruido (typos).
- Etiquetar 300–500 frases reales de beta testers.
- Entrenar TF-IDF + LogReg; guardar **vocabulario** y **vectorizador** versionados.
- Programar **re-entrenamiento semanal** con feedback validado (cron job del worker).

Ejemplos de reglas y prompts

Regex de monto (MXN): `\$\s?([0-9]{1,3}(?:,[0-9]{3})*(?:\.[0-9]{1,2})?)|\b([0-9]+(?:\.[0-9]{1,2})?)\s?(mxn|pesos|$)`

Mapping rápido de alias → categoría: - "uber", "didi", "taxi", "gasolina" → transporte - "netflix", "spotify", "prime" → suscripciones - "soriana", "super", "abarrotes" → comida_super - "tacos", "sushi", "restaurante" → comida_fuera

Prompt (si decides usar LLM pequeño en server) — clasificación:

```
Sistema: Eres un clasificador de gastos. Solo responde con un slug de categoría
de esta lista: [comida_super, comida_fuera, transporte, vivienda, servicios,
salud, educación, ocio, compras, suscripciones, transferencias, otros].
Usuario: "gasté 120 en tacos ayer"
Asistente: comida_fuera
```

Estructura de pantallas (RN)

1. **Onboarding** → explicación + permiso de micrófono.
2. **Home** → botón "Agregar gasto", resumen del mes.
3. **Agregar gasto** → campo texto + micrófono, sugerencia de categoría + confirmación.
4. **Lista de gastos** → filtros por categoría/fecha.
5. **Dashboard** → gráficas y recomendaciones.
6. **Perfil** → meta de ahorro y configuración.

⛳Próximos pasos

1. Inicializar repo monorepo (pnpm para app, poetry para backend) o repos separados.
2. Montar `docker-compose` con api/db/worker/redis/minio.
3. Implementar `POST /expenses` (texto) con pipeline completa.
4. Prototipo de UI de "Agregar gasto" y "Confirmación".
5. Beta cerrada con 10–20 usuarios para alimentar dataset.