🏛 ALETHEIA — Arquitetura Técnica

> **Aletheia** (ἀλήθεια) — "verdade" em grego. Scanner de rótulos alimentares com IA que revela a verdade sobre o que você come.

📑 Índice

Visão Geral

O usuário aponta a câmera do celular para o código de barras de um alimento. Em menos de 5 segundos, recebe:

- **Explicação simples** de cada ingrediente ("como se tivesse 10 anos")
- **Score de saúde** de 0 a 100 (com breakdown visual)
- **Alertas** sobre ingredientes problemáticos (corantes, conservantes, excesso de sódio)
- **Alternativas** mais saudáveis disponíveis na mesma categoria
- **Compatibilidade** com seu perfil alimentar (alergias, dietas, restrições)

Stack Tecnológica

Backend

Componente	Tecnologia	Justificativa
Framework	FastAPI (Python 3.12)	Async nativo, tipagem forte, docs automáticas
		Migrations

ORM	SQLAlchemy 2.0 + Alembic	versionadas, async support
Banco principal	PostgreSQL 16	JSONB para dados flexíveis, full-text search
Cache	Redis 7	Cache de produtos, rate limiting, sessões
Task queue	Celery + Redis broker	Análises IA em background
Auth	JWT (access + refresh tokens)	Stateless, rotação de tokens
Servidor	Uvicorn + Gunicorn	Workers async para alta concorrência

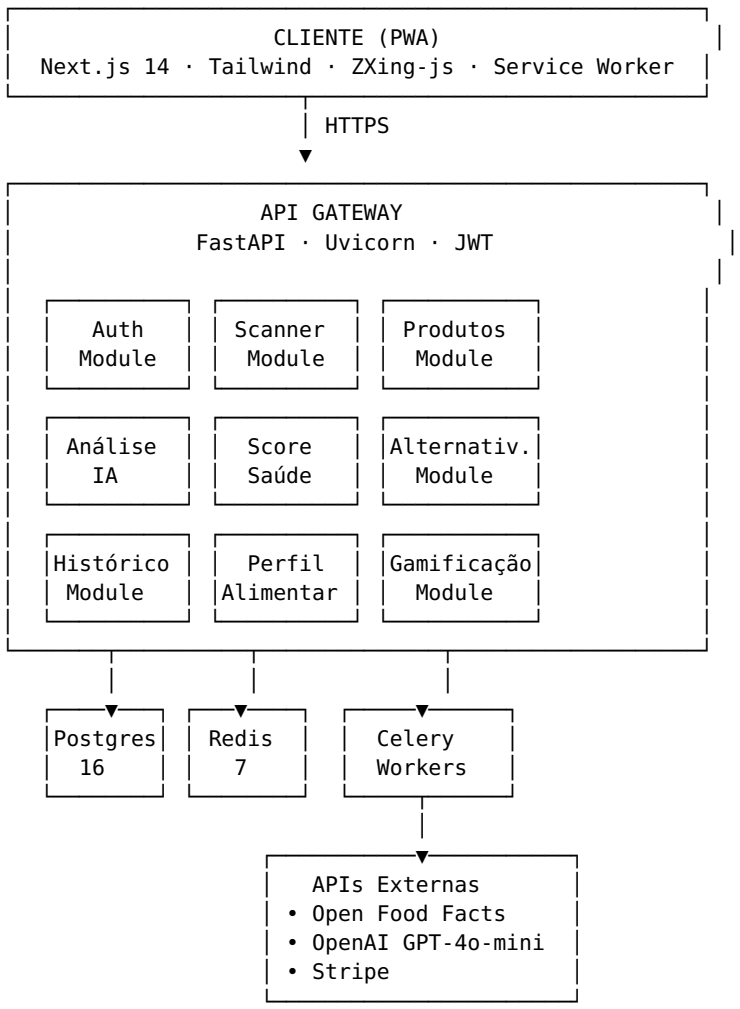
Frontend

Componente	Tecnologia	Justificativa
Framework	Next.js 14 (App Router)	SSR, RSC, PWA-ready
UI	Tailwind CSS + Radix UI	Design system consistente, acessível
Estado	Zustand	Leve, simples, sem boilerplate
Câmera/Barcode	`navigator.mediaDevices` + ZXing-js	Scan direto no browser, sem SDK nativo
HTTP	Axios + React Query (TanStack)	Cache client-side, retry, optimistic updates
PWA	next-pwa (Workbox)	Offline support, install prompt
Animações	Framer Motion	Score gauge, transições suaves

Infra

Componente	Tecnologia
Deploy backend	Railway / Fly.io (ou VPS com Docker)
Deploy frontend	Vercel
CI/CD	GitHub Actions
Monitoramento	Sentry (erros) + Uptime Kuma (status)
Logs	Estruturados com structlog → stdout
Storage	S3-compatible (imagens de produtos)

Arquitetura de Sistema



Módulos

1. Auth Module

Responsabilidade: Registro, login, gerenciamento de sessão.

```
POST /auth/register        → Cria conta (email + senha ou OAuth)
POST /auth/login           → Retorna access_token + refresh_token
POST /auth/refresh         → Renova access_token
POST /auth/forgot-password → Envia email de reset
POST /auth/reset-password  → Aplica nova senha
GET  /auth/me              → Retorna perfil do usuário logado
DELETE /auth/me            → Deleta conta (LGPD)
```

- **Senhas:** bcrypt (cost factor 12)
- **Tokens:** JWT RS256, access_token (15min), refresh_token (30 dias)
- **OAuth:** Google e Apple Sign-In (futuro)
- **Rate limit:** 5 tentativas de login por minuto por IP

2. Scanner Module

Responsabilidade: Decodificação de barcode e orquestração do fluxo de scan.

```
POST /scan                → Recebe barcode, orquestra busca + análise
GET  /scan/{scan_id}      → Retorna resultado completo de um scan
```

Fluxo interno: 1. Recebe barcode (EAN-13/UPC-A) do frontend 2. Verifica rate limit do usuário (créditos) 3. Busca produto no cache Redis → DB local → Open Food Facts 4. Se produto novo, persiste no DB 5. Dispara análise IA (sync se cache hit, async se primeira vez) 6. Retorna resultado consolidado

Barcode no frontend: - ZXing-js para decodificação client-side - Fallback: envio de imagem para Google Cloud Vision API (barcode detection) - Suporte: EAN-13, EAN-8, UPC-A, UPC-E

3. Produtos Module

Responsabilidade: CRUD de produtos, cache e sincronização com Open Food Facts.

```
GET  /products/{barcode}           → Busca produto por barcode
GET  /products/{barcode}/nutrition → Dados nutricionais detalhados
POST /products/report              → Usuário reporta dados incorretos
```

Estratégia de cache (3 camadas):

Camada	TTL	Detalhes
Redis	24h	Hot cache, produtos escaneados recentemente
PostgreSQL	30 dias	Cache persistente, atualizado via cron
Open Food Facts	Sob demanda	Source of truth, fallback

Dados armazenados do produto: - Nome, marca, categoria - Lista de ingredientes (texto original) - Tabela nutricional (por 100g e por porção) - Nutri-Score (A-E) quando disponível - NOVA group (1-4, grau de processamento) - Imagens (frente, ingredientes, nutricional) - Alérgenos declarados

4. Análise IA Module

Responsabilidade: Análise inteligente de ingredientes via GPT-4o-mini.

```
POST /analysis/ingredients   → Analisa lista de ingredientes
GET  /analysis/{analysis_id} → Retorna análise completa
```

Prompt Engineering:

```
        SYSTEM_PROMPT = """
```

```
Você é um nutricionista especialista que explica ingredientes
alimentares de forma simples, como se falasse com alguém sem
formação técnica.

Para cada ingrediente, forneça:
1. Nome popular (se diferente do técnico)
2. O que é e para que serve no produto (1-2 frases)
3. Classificação: ✅ Natural | ⚠️ Atenção | ❌ Evitar
4. Motivo da classificação (1 frase)

Ao final, gere:
- Score de saúde (0-100) com justificativa
- Top 3 ingredientes problemáticos (se houver)
- Resumo em 1 parágrafo para leigo
"""

USER_PROMPT_TEMPLATE = """
Produto: {product_name}
Marca: {brand}
Categoria: {category}
Ingredientes: {ingredients_text}
Tabela nutricional (por 100g): {nutrition_table}
Perfil do usuário: {user_profile}  # alergias, restrições

Analise este produto.
"""
```

Otimizações: - Cache de análises por hash(ingredientes + perfil_usuario) - GPT-4o-mini para custo baixo (~$0.15/1M input tokens) - Structured output (JSON mode) para parsing confiável - Timeout: 10s, retry com exponential backoff - Fallback: análise baseada em regras se IA falhar

Response format (JSON):

```
{
  "ingredients": [
    {
      "name": "Açúcar invertido",
      "popular_name": "Açúcar líquido",
      "explanation": "É açúcar comum dissolvido e processado para
ficar líquido. Usado para adoçar e dar textura.",
      "classification": "warning",
      "reason": "Alto índice glicêmico, contribui para picos de
açúcar no sangue."
    }
  ],
  "health_score": 35,
  "score_breakdown": {
    "naturalness": 20,
    "nutrition": 40,
    "processing": 30,
    "additives": 50
  },
  "problematic_top3": ["Açúcar invertido", "Gordura vegetal
hidrogenada", "Corante caramelo IV"],
  "summary": "Este produto é ultraprocessado com alto teor de
açúcar...",
  "alerts": [
    {"type": "allergen", "message": "Contém glúten (incompatível com
seu perfil)"}
```

```
        ]
    }
```

5. Score de Saúde Module

Responsabilidade: Cálculo do score 0-100, combinando dados nutricionais + análise IA.

Algoritmo de Score:

```python
    def calculate_health_score(product, ai_analysis):
        score = 100

        # 1. Nutri-Score (peso: 25%)
        nutri_penalty = {"a": 0, "b": 5, "c": 15, "d": 25, "e": 35}
        score -= nutri_penalty.get(product.nutri_score, 20) * 0.25

        # 2. NOVA Group - Processamento (peso: 25%)
        nova_penalty = {1: 0, 2: 10, 3: 25, 4: 40}
        score -= nova_penalty.get(product.nova_group, 30) * 0.25

        # 3. Ingredientes problemáticos (peso: 25%)
        problematic_count = len(ai_analysis.problematic_ingredients)
        score -= min(problematic_count * 8, 40) * 0.25

        # 4. Perfil nutricional (peso: 25%)
        # Penaliza excesso de: sódio, açúcar, gordura saturada, gordura
trans
        # Bonifica presença de: fibra, proteína, vitaminas
        nutrition_score =
calculate_nutrition_subscore(product.nutrition)
        score -= (100 - nutrition_score) * 0.25

        return max(0, min(100, round(score)))
```

Visualização: - Gauge circular animado (0-100) - Cores: 🔴 0-30 | 🟠 31-50 | 🟡 51-70 | 🟢 71-100 - Breakdown em 4 categorias com barras horizontais

6. Histórico Module

Responsabilidade: Timeline de scans do usuário, estatísticas e tendências.

```
GET  /history               → Lista scans do usuário (paginado)
GET  /history/stats         → Estatísticas gerais (score médio,
total scans)
GET  /history/trends        → Evolução do score ao longo do tempo
DELETE /history/{scan_id}   → Remove scan do histórico
```

Funcionalidades: - Filtro por período, score range, categoria - Score médio dos últimos 7/30/90 dias - Gráfico de tendência (melhoria ao longo do tempo) - “Seus piores hábitos” (categorias com menor score médio) - Export CSV/PDF (premium)

7. Alternativas Module

Responsabilidade: Sugerir produtos mais saudáveis na mesma categoria.

```
GET /alternatives/{barcode}  → Alternativas para um produto
```

Lógica: 1. Identifica categoria do produto (ex: “biscoito recheado”) 2. Busca produtos da mesma categoria no DB com score > produto atual 3. Ordena por: score DESC, popularidade (nº de scans) DESC 4. Retorna top 5 alternativas com comparativo

Response:

```
        {
          "current_product": {"name": "Biscoito X", "score": 25},
          "alternatives": [
            {
              "name": "Biscoito Integral Y",
              "brand": "Marca Y",
              "score": 72,
              "score_diff": "+47",
              "highlights": ["Sem gordura trans", "Rico em fibras", "Menos
açúcar"],
              "barcode": "7891234567890"
            }
          ]
        }
```

8. Perfil Alimentar Module

Responsabilidade: Preferências, alergias e restrições do usuário.

```
GET  /profile/dietary        → Retorna perfil alimentar
PUT  /profile/dietary        → Atualiza perfil
GET  /profile/dietary/check/{barcode} → Verifica compatibilidade
```

Dados do perfil:

```
        {
          "allergies": ["glúten", "lactose", "amendoim"],
          "intolerances": ["frutose"],
          "diet": "vegetariano",  // null, vegetariano, vegano, low-carb,
keto, etc
          "avoid": ["corantes artificiais", "glutamato monossódico"],
          "goals": ["reduzir açúcar", "mais fibra"],
          "conditions": ["diabetes tipo 2", "hipertensão"]  // premium
        }
```

Impacto no fluxo: - Análise IA recebe perfil como contexto - Alertas personalizados (“⚠ Contém glúten — você é celíaco”) - Score ajustado por relevância pessoal - Alternativas filtradas por compatibilidade

9. Gamificação Module

Responsabilidade: Engajamento via conquistas, streaks e níveis.

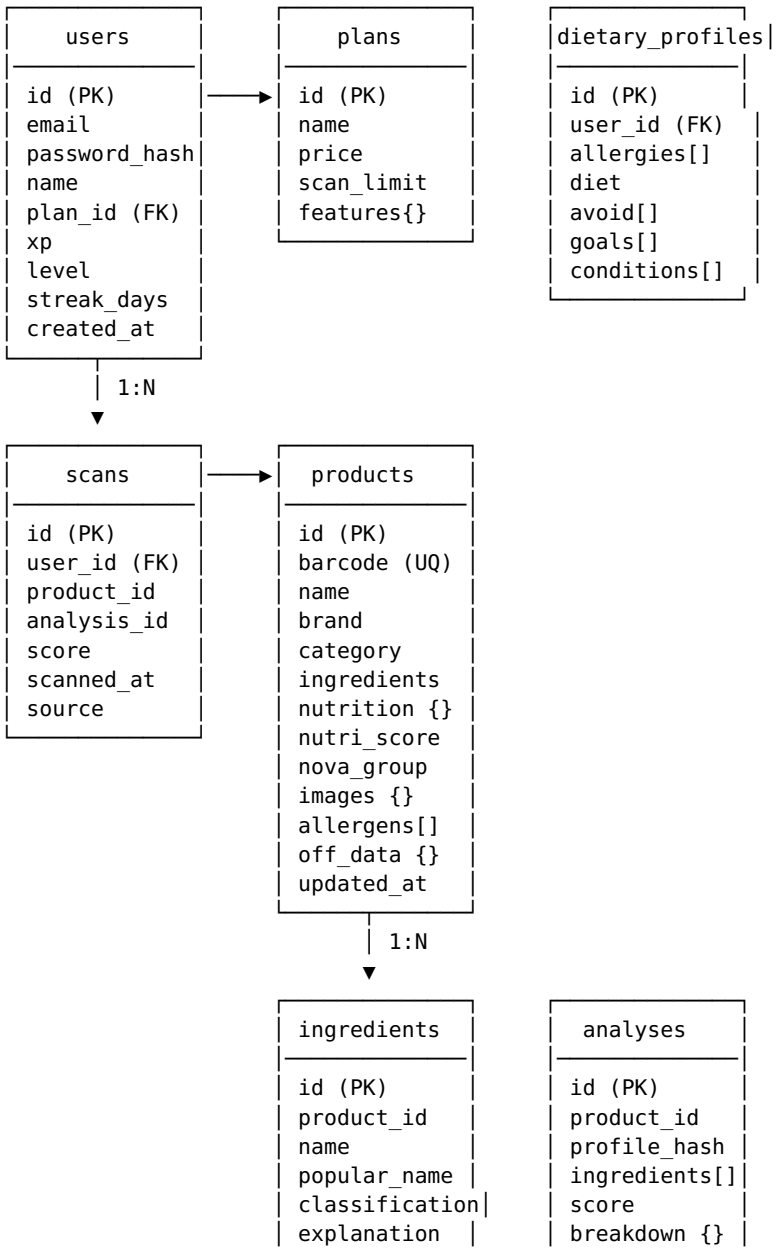
```
GET  /gamification/profile      → XP, nível, conquistas
GET  /gamification/achievements → Lista todas as conquistas
GET  /gamification/leaderboard  → Ranking semanal (premium)
```

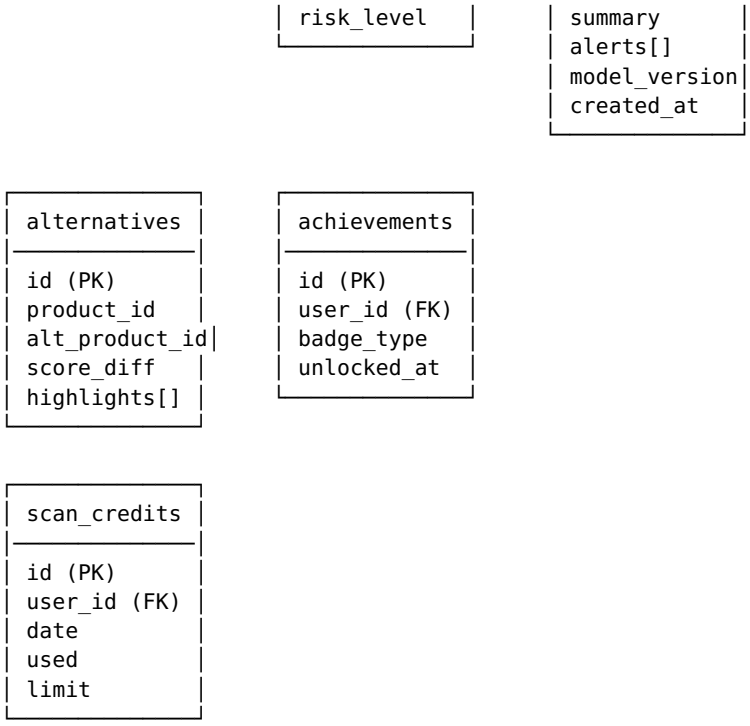
Sistema de XP: | Ação | XP | |—|—| | Scan de produto | +10 | | Primeiro scan do dia | +20 (bônus streak) | | Escolher alternativa saudável | +30 | | Completar perfil alimentar | +50 | | Streak de 7 dias | +100 | | Compartilhar resultado | +15 |

Conquistas (badges): - □ Primeiro Scan - 🔍 Detetive (10 scans) - 🏆 Expert (100 scans) - □ Escolha Saudável (5 alternativas escolhidas) - 🔥 Streak Master (30 dias seguidos) - 📊 Analista (score médio > 70 no mês)

Modelo de Dados

Diagrama ER Simplificado





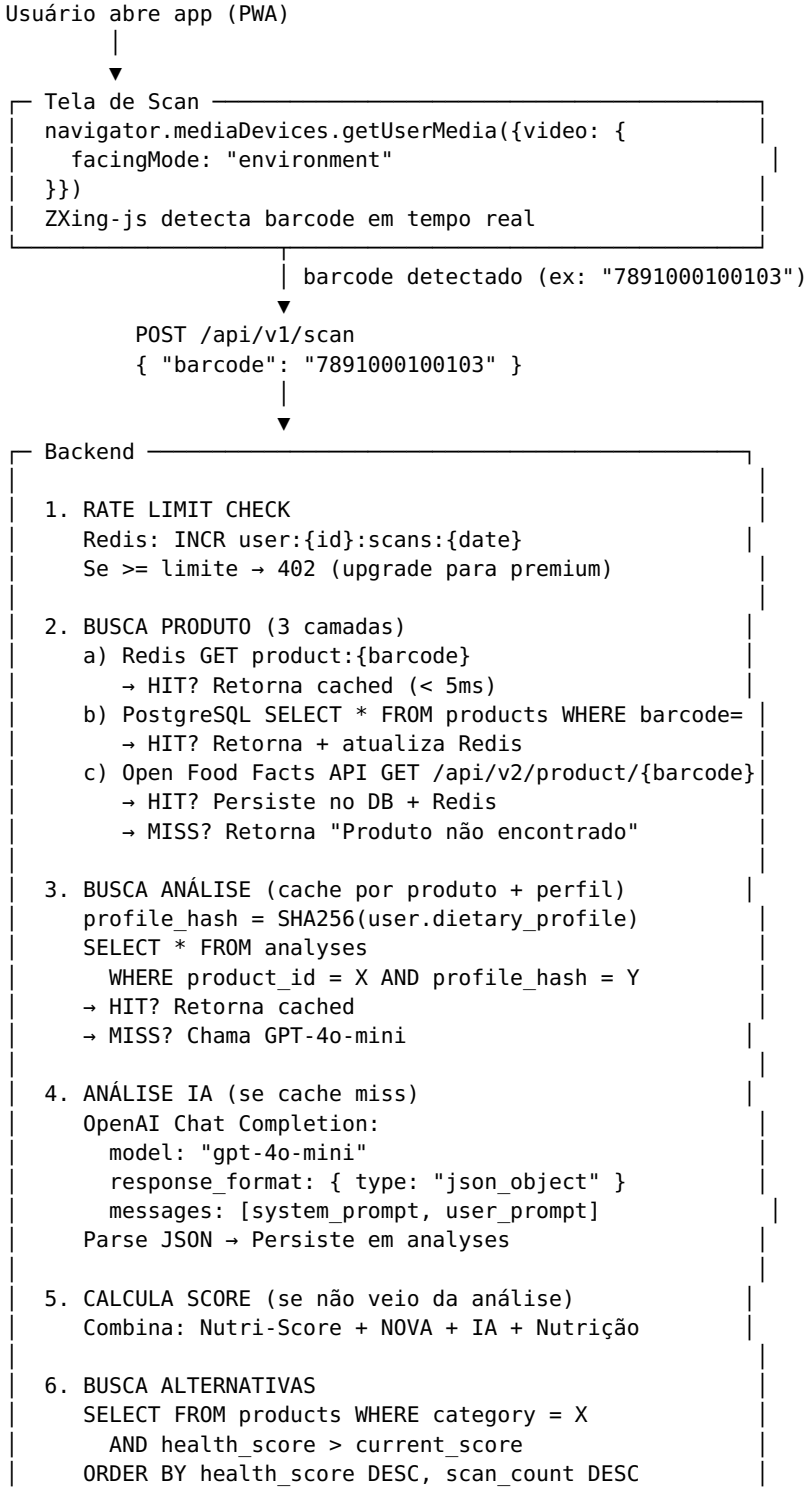
DDL Principais Tabelas

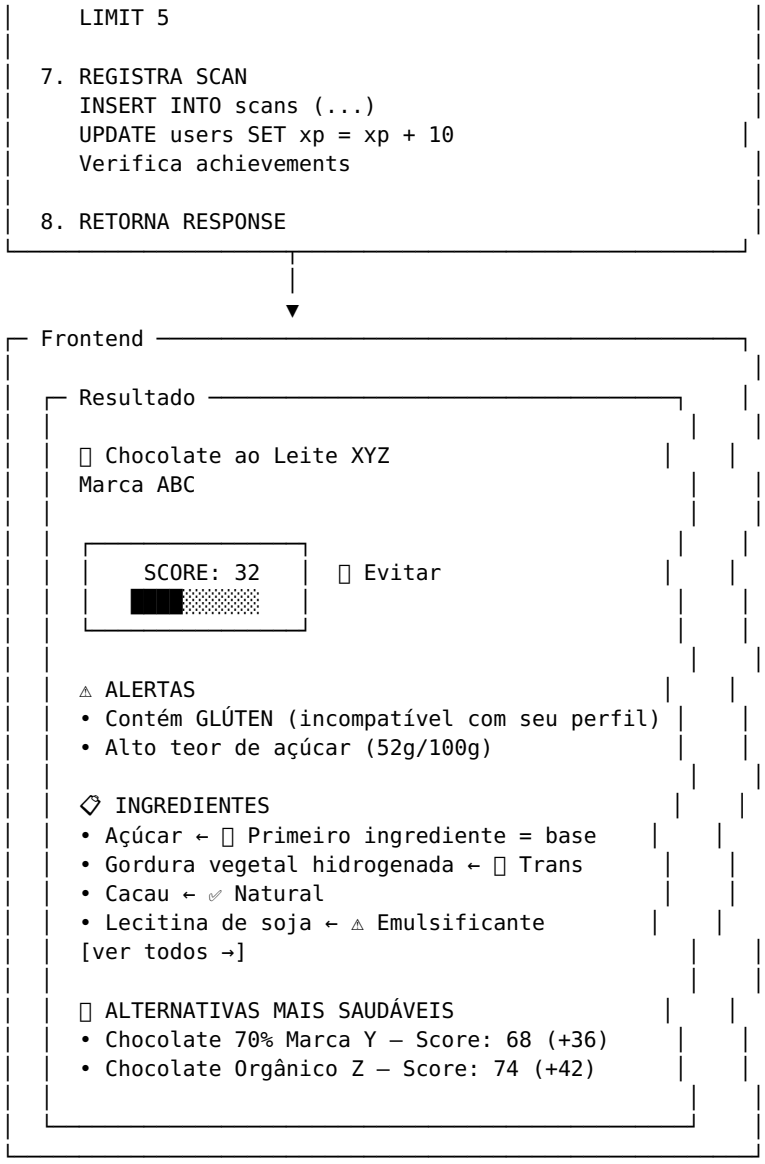
```sql
-- Usuários
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    plan_id INTEGER REFERENCES plans(id) DEFAULT 1,
    xp INTEGER DEFAULT 0,
    level INTEGER DEFAULT 1,
    streak_days INTEGER DEFAULT 0,
    last_scan_date DATE,
    stripe_customer_id VARCHAR(255),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Planos
CREATE TABLE plans (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,          -- 'free', 'premium'
    price_cents INTEGER NOT NULL,        -- 0, 1490 (R$14.90)
    scan_limit_daily INTEGER NOT NULL,   -- 3, -1 (unlimited)
    features JSONB DEFAULT '{}'          -- {"export": true,
"history_full": true}
);

-- Produtos
CREATE TABLE products (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    barcode VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(500),
    brand VARCHAR(200),
```

```
            category VARCHAR(200),
            ingredients_text TEXT,
            nutrition JSONB,            -- {"energy_kcal": 250, "fat": 12,
...}
            nutri_score CHAR(1),        -- A-E
            nova_group SMALLINT,        -- 1-4
            images JSONB,               -- {"front": "url", "ingredients":
"url"}
            allergens TEXT[],
            off_data JSONB,             -- Raw Open Food Facts response
            scan_count INTEGER DEFAULT 0,
            created_at TIMESTAMPTZ DEFAULT NOW(),
            updated_at TIMESTAMPTZ DEFAULT NOW()
        );

        CREATE INDEX idx_products_barcode ON products(barcode);
        CREATE INDEX idx_products_category ON products(category);

        -- Scans
        CREATE TABLE scans (
            id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
            user_id UUID REFERENCES users(id) ON DELETE CASCADE,
            product_id UUID REFERENCES products(id),
            analysis_id UUID REFERENCES analyses(id),
            health_score SMALLINT,
            scanned_at TIMESTAMPTZ DEFAULT NOW()
        );

        CREATE INDEX idx_scans_user_date ON scans(user_id, scanned_at DESC);

        -- Análises IA
        CREATE TABLE analyses (
            id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
            product_id UUID REFERENCES products(id),
            profile_hash VARCHAR(64),    -- SHA-256 do perfil alimentar
(para cache)
            ingredients_analysis JSONB,  -- Array de análises por
ingrediente
            health_score SMALLINT,
            score_breakdown JSONB,       -- {"naturalness": 20, "nutrition":
40, ...}
            problematic_top3 TEXT[],
            summary TEXT,
            alerts JSONB,
            model_version VARCHAR(50),   -- "gpt-4o-mini-2024-07-18"
            tokens_used INTEGER,
            created_at TIMESTAMPTZ DEFAULT NOW()
        );

        CREATE INDEX idx_analyses_product_profile ON analyses(product_id,
profile_hash);

        -- Perfis Alimentares
        CREATE TABLE dietary_profiles (
            id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
            user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
            allergies TEXT[] DEFAULT '{}',
            intolerances TEXT[] DEFAULT '{}',
            diet VARCHAR(50),
            avoid TEXT[] DEFAULT '{}',
```

```
            goals TEXT[] DEFAULT '{}',
            conditions TEXT[] DEFAULT '{}',
            updated_at TIMESTAMPTZ DEFAULT NOW()
        );
```

Fluxo Principal





Sequência temporal (target < 5s)

Etapa	Tempo (cache hit)	Tempo (cache miss)
Decodificação barcode (client)	~200ms	~200ms
Request → Backend	~100ms	~100ms
Rate limit check (Redis)	~5ms	~5ms
Busca produto	~5ms (Redis)	~800ms (OFF API)
Busca/gera análise IA	~5ms (cache)	~2500ms (GPT)
Calcula score	~10ms	~10ms
Busca alternativas	~50ms	~50ms
Registra scan + XP	~20ms	~20ms

Response → Frontend	~100ms	~100ms
Render resultado	~200ms	~200ms
TOTAL	**~700ms** ✅	**~4000ms** ✅

APIs Externas

Open Food Facts

- **URL:** `https://world.openfoodfacts.org/api/v2/product/{barcode}.json`
- **Custo:** Gratuita, open source
- **Rate limit:** 100 req/min (ser gentil)
- **Cobertura:** ~3M produtos, boa cobertura Brasil
- **User-Agent obrigatório:** `Aletheia/1.0 (contato@aletheia.app)`
- **Fallback:** Se produto não encontrado, permitir cadastro manual (v2.0)

OpenAI GPT-4o-mini

- **Endpoint:** `POST https://api.openai.com/v1/chat/completions`
- **Modelo:** `gpt-4o-mini`
- **Custo estimado:**
  - Input: ~500 tokens/scan × $0.15/1M = $0.000075/scan
  - Output: ~800 tokens/scan × $0.60/1M = $0.00048/scan
  - **Total: ~$0.00055/scan ≈ R$0.003/scan**
  - 10K scans/dia = ~R$30/dia
- **Timeout:** 10 segundos
- **Retry:** 3x com exponential backoff (1s, 2s, 4s)
- **Fallback:** Análise baseada em regras (lista de ingredientes conhecidos)

Stripe

- **Uso:** Assinaturas (Checkout + Customer Portal)
- **Webhooks:** `invoice.paid`, `customer.subscription.updated`, `customer.subscription.deleted`
- **Planos:**
  - Free: R$0 (3 scans/dia)
  - Premium: R$14,90/mês (ilimitado + features extras)

Google Cloud Vision (fallback barcode)

- **Uso:** Apenas quando ZXing-js falha na decodificação client-side
- **Endpoint:** `POST https://vision.googleapis.com/v1/images:annotate`
- **Custo:** $1.50/1K imagens (primeiras 1K/mês grátis)
- **Alternativa free:** QuaggaJS como segundo decoder client-side

---

PWA & Câmera

Service Worker (next-pwa)

```
// next.config.js
const withPWA = require('next-pwa')({
  dest: 'public',
  register: true,
  skipWaiting: true,
  runtimeCaching: [
    {
      urlPattern:
/^https:\/\/api\.aletheia\.app\/api\/v1\/products\/.*/,
      handler: 'StaleWhileRevalidate',
      options: {
        cacheName: 'product-cache',
        expiration: { maxEntries: 200, maxAgeSeconds: 86400 }
      }
    }
  ]
});
```

Câmera & Barcode Scanner

```
// hooks/useBarcodeScan.ts
import { BrowserMultiFormatReader } from '@zxing/library';

export function useBarcodeScan() {
  const videoRef = useRef<HTMLVideoElement>(null);
  const readerRef = useRef(new BrowserMultiFormatReader());

  const startScan = async () => {
    const stream = await navigator.mediaDevices.getUserMedia({
      video: {
        facingMode: 'environment',  // câmera traseira
        width: { ideal: 1280 },
        height: { ideal: 720 },
      }
    });

    videoRef.current!.srcObject = stream;

    readerRef.current.decodeFromVideoDevice(
      undefined,  // usa device padrão
      videoRef.current!,
      (result, error) => {
        if (result) {
          // Vibra para feedback
          navigator.vibrate?.(200);
          // Envia barcode para API
          onBarcodeDetected(result.getText());
        }
      }
    );
  };

  return { videoRef, startScan, stopScan };
}
```

Manifest (PWA)

```
        {
          "name": "Aletheia - Scanner de Rótulos",
          "short_name": "Aletheia",
          "description": "Descubra a verdade sobre o que você come",
          "start_url": "/",
          "display": "standalone",
          "orientation": "portrait",
          "theme_color": "#16A34A",
          "background_color": "#FFFFFF",
          "icons": [
            { "src": "/icons/icon-192.png", "sizes": "192x192", "type":
"image/png" },
            { "src": "/icons/icon-512.png", "sizes": "512x512", "type":
"image/png" }
          ]
        }
```

Monetização Técnica

Planos

Feature	Free	Premium (R$14,90/mês)
Scans por dia	3	Ilimitado
Histórico	Últimos 7 dias	Completo
Análise IA	Básica	Detalhada + perfil
Alternativas	Top 2	Top 5 + comparativo
Perfil alimentar	Alergias apenas	Completo (condições)
Export (CSV/PDF)	✗	✓
Leaderboard	✗	✓
Sem anúncios	✗	✓

Fluxo Stripe

```
Usuário clica "Upgrade Premium"
        |
        ▼
POST /billing/checkout-session
  → Stripe Checkout Session (mode: subscription)
  → Redirect para Stripe hosted page
        |
        ▼
Stripe processa pagamento
        |
        ▼
Webhook: invoice.paid
  → Backend atualiza user.plan_id = 2
  → Redis: SET user:{id}:plan premium
        |
        ▼
```

```
Usuário retorna ao app → plano ativo
```

Sistema de Créditos

```python
async def check_scan_credits(user_id: str) -> bool:
    today = date.today().isoformat()
    key = f"credits:{user_id}:{today}"

    # Usuário premium → sempre permitido
    if await is_premium(user_id):
        return True

    used = await redis.get(key)
    if used and int(used) >= DAILY_FREE_LIMIT:  # 3
        return False

    await redis.incr(key)
    await redis.expire(key, 86400)  # expira em 24h
    return True
```

Performance

Metas

Métrica	Target	Estratégia
Scan → resultado	< 5s (cold) / < 1s (cached)	Cache 3 camadas
TTFB (first byte)	< 200ms	Edge deploy (Vercel)
LCP	< 2.5s	SSR + lazy load imagens
Bundle size	< 150KB (gzipped)	Tree shaking, dynamic imports
Lighthouse PWA	> 90	Service worker, manifest, HTTPS
API p95 latency	< 500ms (cached)	Redis, connection pooling
Uptime	99.9%	Health checks, auto-restart

Estratégias de Cache

```
┌──────────────────────────────────────────────┐
│           Cache Architecture                 │
│                                              │
│  Client (React Query)                        │
│    └─ staleTime: 5min                        │
│    └─ Produtos escaneados ficam em memória   │
│                                              │
│  Service Worker (Workbox)                    │
│    └─ StaleWhileRevalidate para /products    │
│    └─ CacheFirst para imagens estáticas      │
│                                              │
```

```
│  Redis (Server)                                  │
│    └─ product:{barcode} → TTL 24h                │
│    └─ analysis:{product_id}:{hash} → TTL 7d      │
│    └─ user:{id}:credits:{date} → TTL 24h         │
│                                                  │
│  PostgreSQL                                      │
│    └─ Source of truth, updated via cron          │
│    └─ Produtos atualizados a cada 30 dias        │
└──────────────────────────────────────────────────┘
```

Otimizações Backend

- **Connection pooling:** SQLAlchemy async pool (min=5, max=20)
- **Bulk operations:** Batch insert para alternativas
- **Índices:** barcode (B-tree), category (B-tree), scans por user+date
- **Async everywhere:** FastAPI + httpx (Open Food Facts) + asyncpg
- **Streaming response:** Para análises longas, usar SSE (Server-Sent Events)

Design & UX

Identidade Visual

Elemento	Especificação
Nome	Aletheia (ἀλήθεια = verdade)
Logo	Olho grego estilizado (Nazar/Mati) com íris em forma de barcode
Cores primárias	Verde `#16A34A` (saúde) + Branco `#FFFFFF` (clean)
Cores secundárias	Cinza `#6B7280` (texto) + Verde claro `#DCFCE7` (backgrounds)
Cores de score	🔴 `#EF4444` · 🟠 `#F97316` · 🟡 `#EAB308` · 🟢 `#22C55E`
Tipografia	Inter (UI) + Plus Jakarta Sans (headings)
Ícones	Lucide Icons (consistente, leve)
Bordas	`rounded-xl` (16px), sombras suaves
Espaçamento	Grid 8px, padding generoso

Telas Principais

```
1. SPLASH / ONBOARDING
   - Logo Aletheia (olho grego animado)
   - "Descubra a verdade sobre o que você come"
   - 3 slides: Scan → Entenda → Escolha melhor
   - CTA: "Começar" → setup perfil alimentar

2. HOME
```

```
   - Header: logo + streak 🔥 + XP bar
   - Botão central grande: "📸 Escanear"
   - Últimos scans (horizontal scroll)
   - Score médio da semana (mini gauge)
   - Tip do dia (IA)

3. SCANNER
   - Câmera fullscreen com overlay
   - Guia visual: "Aponte para o código de barras"
   - Auto-detect + vibração
   - Loading: animação do olho "analisando"

4. RESULTADO
   - Score gauge animado (destaque principal)
   - Alertas personalizados (cards vermelhos/amarelos)
   - Lista de ingredientes (expandível, com ícones)
   - Alternativas (cards horizontais)
   - Botões: Salvar | Compartilhar | Escanear outro

5. HISTÓRICO
   - Timeline vertical com mini scores
   - Filtros: período, categoria, score
   - Gráfico de tendência (line chart)

6. PERFIL
   - Dados pessoais
   - Perfil alimentar (alergias, dieta, goals)
   - Plano (free/premium)
   - Gamificação (nível, badges, streak)
   - Configurações
```

Micro-interações

- **Scan detectado:** Vibração + flash verde + som sutil
- **Score reveal:** Animação circular de 0 até valor final (1.5s)
- **Ingrediente tap:** Expande com animação suave (Framer Motion)
- **Achievement unlocked:** Toast animado com confetti
- **Pull to refresh:** Animação do olho grego piscando

Roadmap

MVP — Semanas 1-3

Semana 1: Infraestrutura + **Scanner** - [ ] Setup monorepo (turborepo ou pasta separada) - [ ] Backend: FastAPI boilerplate, auth JWT, migrations - [ ] Frontend: Next.js 14, PWA setup, layout base - [ ] Scanner: câmera + ZXing-js funcionando - [ ] Integração Open Food Facts (busca básica)

Semana 2: IA + **Core Features** - [ ] Análise IA com GPT-4o-mini - [ ] Score de saúde (algoritmo v1) - [ ] Tela de resultado completa - [ ] Cache Redis para produtos + análises - [ ] Histórico básico (lista de scans)

Semana 3: Polish + Deploy - [ ] Design system (cores, tipografia, componentes) - [ ] Rate limiting (3 scans/dia free) - [ ] Alternativas básicas - [ ] Error handling e loading states - [ ] Deploy: Vercel (front) + Railway (back) - [ ] Testes E2E dos fluxos principais

Entregáveis MVP: - PWA funcional no celular - Scan → resultado com score + ingredientes explicados - 3 scans grátis por dia - Histórico dos últimos 7 dias

v1.0 — Semanas 4-6

- ☐ Perfil alimentar completo
- ☐ Alertas personalizados baseados no perfil
- ☐ Stripe integration (premium)
- ☐ Gamificação (XP, streaks, badges)
- ☐ Onboarding flow
- ☐ Push notifications (Web Push API)
- ☐ Offline mode (scans salvos localmente)

v2.0 — Semanas 7-10

- ☐ Compartilhar resultado (social cards)
- ☐ Comparar 2 produtos lado a lado
- ☐ Leaderboard semanal
- ☐ OCR de ingredientes (foto da lista quando sem barcode)
- ☐ Cadastro de produtos por usuários
- ☐ API pública para desenvolvedores
- ☐ Internacionalização (PT-BR, EN, ES)

v3.0 — Futuro

- ☐ App nativo (React Native ou Capacitor)
- ☐ Integração com supermercados (preços)
- ☐ Scan de cardápios de restaurantes
- ☐ Diário alimentar com score diário
- ☐ Recomendações de receitas saudáveis
- ☐ Integração com Apple Health / Google Fit

Estrutura de Diretórios

```
aletheia/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py                # FastAPI app factory
│   │   ├── config.py              # Settings (pydantic-settings)
│   │   ├── database.py            # SQLAlchemy async engine
│   │   ├── dependencies.py        # Shared deps (get_db,
get_current_user)
│   │   ├── models/                # SQLAlchemy models
```

```
│   │   │   ├── user.py
│   │   │   ├── product.py
│   │   │   ├── scan.py
│   │   │   ├── analysis.py
│   │   │   ├── dietary_profile.py
│   │   │   └── plan.py
│   │   ├── schemas/                # Pydantic schemas
│   │   │   ├── user.py
│   │   │   ├── product.py
│   │   │   ├── scan.py
│   │   │   └── analysis.py
│   │   ├── routers/                # API routes
│   │   │   ├── auth.py
│   │   │   ├── scan.py
│   │   │   ├── products.py
│   │   │   ├── analysis.py
│   │   │   ├── history.py
│   │   │   ├── alternatives.py
│   │   │   ├── profile.py
│   │   │   ├── gamification.py
│   │   │   └── billing.py
│   │   ├── services/               # Business logic
│   │   │   ├── auth_service.py
│   │   │   ├── scan_service.py
│   │   │   ├── product_service.py
│   │   │   ├── ai_service.py
│   │   │   ├── score_service.py
│   │   │   ├── alternatives_service.py
│   │   │   ├── credits_service.py
│   │   │   └── gamification_service.py
│   │   ├── integrations/           # External APIs
│   │   │   ├── open_food_facts.py
│   │   │   ├── openai_client.py
│   │   │   └── stripe_client.py
│   │   └── utils/
│   │       ├── cache.py
│   │       ├── security.py
│   │       └── prompts.py
│   ├── alembic/                    # Migrations
│   ├── tests/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── pyproject.toml
│
├── frontend/
│   ├── src/
│   │   ├── app/                    # Next.js App Router
│   │   │   ├── layout.tsx
│   │   │   ├── page.tsx            # Home
│   │   │   ├── scan/page.tsx       # Scanner
│   │   │   ├── result/[id]/page.tsx
│   │   │   ├── history/page.tsx
│   │   │   ├── profile/page.tsx
│   │   │   └── premium/page.tsx
│   │   ├── components/
│   │   │   ├── ui/                 # Design system
│   │   │   ├── scanner/
│   │   │   │   ├── CameraView.tsx
│   │   │   │   └── BarcodeOverlay.tsx
│   │   │   ├── result/
```

```
│   │   │   │   ├── ScoreGauge.tsx
│   │   │   │   ├── IngredientList.tsx
│   │   │   │   ├── AlertCards.tsx
│   │   │   │   └── AlternativeCards.tsx
│   │   │   └── gamification/
│   │   │       ├── XPBar.tsx
│   │   │       └── BadgeGrid.tsx
│   │   ├── hooks/
│   │   │   ├── useBarcodeScan.ts
│   │   │   ├── useAuth.ts
│   │   │   └── useScan.ts
│   │   ├── lib/
│   │   │   ├── api.ts              # Axios instance
│   │   │   └── utils.ts
│   │   └── stores/
│   │       ├── authStore.ts        # Zustand
│   │       └── scanStore.ts
│   ├── public/
│   │   ├── manifest.json
│   │   ├── sw.js
│   │   └── icons/
│   ├── next.config.js
│   ├── tailwind.config.ts
│   └── package.json
│
├── docs/
│   ├── ARQUITETURA-TECNICA.md      # Este arquivo
│   ├── API.md                      # Documentação da API
│   └── DEPLOY.md                   # Guia de deploy
│
├── docker-compose.yml              # PostgreSQL + Redis + Backend
├── .env.example
└── README.md
```

Estimativa de Custos (10K usuários ativos)

Serviço	Custo/mês
Vercel (frontend)	Free (hobby) ou $20 (pro)
Railway (backend + DB + Redis)	~$20-40
OpenAI GPT-4o-mini (~30K scans/mês)	~R$90 (~$17)
Stripe (2.5% + fees)	Variável
Domínio	~R$40/ano
Total estimado	**~R$250-400/mês**

Break-even: ~25-30 assinantes premium (R$14,90 × 30 = R$447)

> *"Aletheia — porque você merece saber a verdade sobre o que come."* 👁️