Plan Maestro y Arquitectura de Software

Proyecto: Wara Citas

Versión 1.0 - Documento Estratégico

Documento preparado para el equipo de desarrollo por ahora Juan Miguel

10 de agosto de 2025

Índice

1. Visión y Estrategia del Producto

1.1. Declaración de Visión

Convertir a **Chispa Cubana** en la plataforma de citas por excelencia para la comunidad cubana global. Fomentaremos conexiones auténticas y seguras mediante tecnología de punta, un profundo entendimiento cultural y un compromiso inquebrantable con la confianza del usuario.

1.2. Propuesta Única de Valor (PUV)

Çonexiones con tu misma sazón." No somos un clon de Tinder con un logo cubano. Somos una experiencia curada que entiende los matices culturales, el lenguaje y los valores que unen a los cubanos, sin importar en qué parte del mundo se encuentren.

1.3. Modelo de Monetización Avanzado

El modelo *Freemium* es la base para la adquisición de usuarios. La monetización se estructura en tres niveles:

1. **Chispa Básico (Gratis):** Funcionalidad esencial para garantizar una experiencia completa y la creación de una masa crítica de usuarios.
2. **Chispa Plus (Suscripción):** Un tier que ofrece *conveniencia*.
   - Likes ilimitados, Rewind, Passport, Ver a quién le gustas.
3. **Chispa Platinum (Suscripción Premium):** Un tier que ofrece *ventaja competitiva*.
   - Todas las funciones de Plus.
   - **Prioridad en Likes:** Tus "likes"se muestran antes a otros usuarios.
   - **Ver Likes Enviados:** Un historial de los perfiles a los que has dado like en los últimos 7 días.
   - **Mensaje antes de Match:** Posibilidad de adjuntar un breve mensaje a una "Super Chispa".
4. **Consumibles (A la Carta):** Compras de "Boostsz "Super Chispas".

2. Arquitectura y Stack Tecnológico

La arquitectura está diseñada para **escalabilidad horizontal, alto rendimiento y mantenibilidad**.

2.1. Stack Tecnológico Recomendado

- **Frontend (Móvil): React Native**. Para un desarrollo unificado en iOS y Android.
- **Backend: Node.js con TypeScript** y el framework **NestJS**. NestJS proporciona una arquitectura modular y escalable (similar a Angular), ideal para proyectos grandes.
- **Base de Datos Principal: PostgreSQL**. Por su robustez, extensibilidad y potente soporte para operaciones geoespaciales a través de la extensión **PostGIS**.
- **Base de Datos en Memoria (Caché): Redis**. Para almacenar en caché perfiles de usuario, sesiones y limitar tasas de peticiones (rate limiting).

- **Mensajería en Tiempo Real (Chat): WebSockets**. Implementado a través de un servicio como **Socket.IO** o un servicio gestionado como **Ably** para reducir la complejidad.
- **Cola de Mensajes (Message Broker): RabbitMQ** o **AWS SQS**. Para procesar tareas en segundo plano de forma asíncrona (ej: notificaciones, actualización de rankings).
- **Infraestructura y Despliegue (Cloud): Amazon Web Services (AWS)** o **Google Cloud Platform (GCP)**.
  - **Contenerización: Docker**.
  - **Orquestación: Kubernetes (EKS en AWS, GKE en GCP)**. Para gestionar y escalar los microservicios de forma automática.
- **Servicios de Terceros (Enfoque Híbrido):**
  - **Autenticación: Firebase Authentication**. Acelera el desarrollo inicial y ofrece seguridad robusta (OAuth, verificación por SMS).
  - **Almacenamiento de Archivos: AWS S3** o **Google Cloud Storage**. Para almacenar fotos y videos de perfil.
  - **Notificaciones Push: Firebase Cloud Messaging (FCM)**.
  - **Email Transaccional: AWS SES** o **SendGrid**.

3. Modelo de Datos Detallado (PostgreSQL)

3.1. Diagrama de Entidad-Relación (Simplificado)

Imagina un diagrama donde ‘users‘ es la tabla central. ‘photos‘, ‘interests‘, ‘swipes‘ y ‘blocks‘ se relacionan con ‘users‘. ‘matches‘ conecta a dos usuarios, y ‘messages‘ se relaciona con ‘matches‘.

3.2. Definición de Tablas Principales

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    firebase_uid VARCHAR(255) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    name VARCHAR(100),
    birthdate DATE,
    gender VARCHAR(50),
    bio TEXT,
    job_title VARCHAR(100),
    company VARCHAR(100),
    -- PostGIS para localizacion eficiente
    location GEOGRAPHY(POINT, 4326),
    is_verified BOOLEAN DEFAULT FALSE,
    last_active TIMESTAMPTZ DEFAULT NOW(),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    -- Configuracion de preferencias
    pref_gender VARCHAR(50)[] DEFAULT ARRAY['todos'],
    pref_age_min INT DEFAULT 18,
    pref_age_max INT DEFAULT 55,
    pref_distance_km INT DEFAULT 50
);

-- Indice espacial para busquedas geograficas rapidas
CREATE INDEX users_location_idx ON users USING GIST (location);

```

```sql
CREATE TABLE user_rankings (
    user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
    -- Glicko-2 es superior a ELO
    glicko_rating REAL DEFAULT 1500.0,
    glicko_deviation REAL DEFAULT 350.0,
    glicko_volatility REAL DEFAULT 0.06,
    last_updated TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE swipes (
    swiper_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    swiped_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    direction VARCHAR(10) NOT NULL, -- 'right' o 'left'
    created_at TIMESTAMPTZ DEFAULT NOW(),
    PRIMARY KEY (swiper_id, swiped_id)
);

CREATE TABLE matches (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user1_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    user2_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    -- Restriccion para evitar duplicados (user1, user2) vs (user2, user1)
    UNIQUE (LEAST(user1_id, user2_id), GREATEST(user1_id, user2_id))
);

CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    match_id UUID NOT NULL REFERENCES matches(id) ON DELETE CASCADE,
    sender_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

4. Algoritmos de Emparejamiento: El Núcleo Inteligente

El algoritmo es un sistema de puntuación compuesto, diseñado para ser justo, eficiente y promover matches de alta calidad.

4.1. Pilar 1: Ranking de Atractivo (Sistema Glicko-2)

Abandonamos el sistema ELO simple en favor de **Glicko-2**, que modela el .ªtractivo"de un perfil de forma más precisa al incluir una **desviación** (qué tan seguro estamos del ranking) y una **volatilidad** (qué tan consistente es el ranking).

4.1.1. Lógica de Actualización (Tarea Asíncrona)

Cuando el 'Usuario A' desliza al 'Usuario B', se encola una tarea. Un worker procesa esta tarea:

1. Carga los ratings Glicko-2 de A y B desde la tabla user-rankings.
2. Trata el swipe como un "juego". Si es "like", A "ganaz B "pierde". Si es "nope", no se considera un juego para no penalizar a los usuarios selectivos.
3. Calcula los nuevos ratings, desviaciones y volatilidades para ambos usuarios usando las formulas de Glicko-2.
4. Actualiza la tabla 'user-rankings'.

4.2. Pilar 2: Descubrimiento de Perfiles (La Pila de Swipes)

Esta es la función principal que se ejecuta cada vez que un usuario abre la app o necesita más perfiles.

4.2.1. Paso 1: Búsqueda Geoespacial con PostGIS

Se ejecuta una consulta altamente eficiente para encontrar candidatos iniciales.

```
SELECT id, location, last_active
FROM users
WHERE ST_DWithin(
    location,
    (SELECT location FROM users WHERE id = :current_user_id),
    :pref_distance_km * 1000 -- Distancia en metros
)
AND id != :current_user_id
-- Aqui se anadirian filtros de edad y genero
LIMIT 500; -- Limitar el conjunto inicial de candidatos
```

4.2.2. Paso 2: Exclusión y Filtrado en el Backend

El servicio de backend recibe la lista de IDs de la base de datos y realiza filtros adicionales que son complejos para SQL:

1. **Excluir Swipes Previos:** Elimina los IDs que ya existen en la tabla ‘swipes‘ para el usuario actual.
2. **Excluir Matches Existentes:** Elimina los IDs con los que ya existe un match.
3. **Excluir Usuarios Bloqueados.**

4.2.3. Paso 3: Puntuación Compuesta y Ranking Final

Para cada perfil candidato restante (‘C‘), se calcula un ‘score‘ final.

$$Score(C) = w_1 \cdot S_{rank} + w_2 \cdot S_{dist} + w_3 \cdot S_{act} + w_4 \cdot S_{int} \tag{1}$$

Donde w_n son pesos configurables para ajustar el algoritmo.

- S_{rank} **(Puntuación de Ranking):** Basado en la proximidad de los ratings Glicko-2. Un valor más alto si los ratings son similares.

$$S_{rank} = 1 - \frac{|\text{rating}(C) - \text{rating}(\text{Yo})|}{R_{max}} \tag{2}$$

- S_{dist} **(Puntuación de Distancia):** Normaliza la distancia para que los perfiles más cercanos obtengan una puntuación más alta.

$$S_{dist} = 1 - \frac{\text{distancia}(C, \text{Yo})}{\text{pref\_distancia\_km}} \tag{3}$$

- S_{act} **(Puntuación de Actividad):** Penaliza la inactividad.

$$S_{act} = e^{-0{,}1 \cdot \text{días\_desde\_última\_actividad}(C)} \tag{4}$$

- S_{int} **(Puntuación de Intereses):** Basado en la superposición de intereses.

$$S_{int} = \frac{|\text{intereses}(C) \cap \text{intereses}(\text{Yo})|}{|\text{intereses}(C) \cup \text{intereses}(\text{Yo})|} \tag{5}$$

Finalmente, la lista de perfiles candidatos se ordena de mayor a menor ‘Score(C)‘ y se envía al frontend.

5. Plan de Desarrollo por Fases (Roadmap)

5.1. Fase 1: Configuración e Infraestructura (Sprints 0-1, 2 Semanas)

- Configuración de la cuenta de AWS/GCP.
- Definición de la infraestructura como código (Terraform).
- Creación del esqueleto del proyecto en NestJS.
- Configuración del pipeline de CI/CD (GitHub Actions) para despliegue automático en un entorno de *staging*.

5.2. Fase 2: Núcleo de Autenticación y Perfiles (Sprints 2-4, 3 Semanas)

- Integración con Firebase Authentication.
- Flujo de registro, inicio de sesión y creación/edición de perfiles.
- Lógica para la subida de imágenes a AWS S3.

5.3. Fase 3: El Corazón de la App (Sprints 5-8, 4 Semanas)

- Implementación del Deck de Swipes en el frontend.
- Creación del endpoint en el backend para el descubrimiento de perfiles (Algoritmo v1).
- Implementación de la lógica de swipes y creación de matches.
- Worker asíncrono para la actualización de rankings Glicko-2.

5.4. Fase 4: Comunicación y Monetización (Sprints 9-11, 3 Semanas)

- Implementación del chat en tiempo real con WebSockets.
- Integración con RevenueCat para gestionar suscripciones de iOS y Android.
- Lógica de backend para restringir/habilitar funcionalidades premium.

5.5. Fase 5: Pruebas y Lanzamiento (Sprint 12, 2 Semanas)

- Pruebas de carga y estrés (Load Testing) con herramientas como k6.
- Beta cerrada con un grupo de usuarios seleccionados.
- Publicación en App Store y Google Play.

6. Roadmap Post-Lanzamiento (Evolución)

- **Q1 Post-Lanzamiento:**
  - Perfiles de video (cortos de 10 segundos).
  - Ajuste y optimización de los pesos del algoritmo (w_n) basado en datos reales de matches.
  - Panel de administración básico.
- **Q2 Post-Lanzamiento:**

- Función de .Eventos": Promocionar eventos de la comunidad cubana y ver qué otros usuarios de la app asistirán.
- Integración con Spotify (canciones favoritas en el perfil).
- Filtros de búsqueda avanzados (educación, intenciones, etc.).