PlaylistAI

Playlists inteligentes de YouTube curadas por IA

Documento de Alcance y Plan de Desarrollo

Febrero 2026

Stack: **React Native + NestJS + GraphQL + PostgreSQL + Docker**

1. Vision del Producto

PlaylistAI es una aplicacion mobile que permite a los usuarios crear playlists personalizadas de YouTube a traves de conversaciones con IA. El usuario describe lo que quiere aprender o ver, la IA hace preguntas de contexto, busca los mejores videos disponibles y arma una playlist curada que se almacena en la app y se reproduce con el player embebido de YouTube.

El diferenciador principal es la experiencia conversacional: en lugar de buscar manualmente entre millones de videos, el usuario dialoga con la IA y obtiene una seleccion optimizada para su objetivo, nivel y tiempo disponible.

---

2. Problema que Resuelve

- YouTube tiene contenido para aprender cualquier cosa, pero encontrar los videos correctos y armar una secuencia coherente es tedioso y consume tiempo.
- Las playlists existentes en YouTube son genericas, no se adaptan al nivel ni objetivos del usuario.
- No existe una herramienta que combine busqueda inteligente + curacion por IA + organizacion personalizada de contenido de YouTube.
- Las playlists de YouTube no permiten agregar notas, marcar progreso, ni trackear aprendizaje.

---

3. Alcance Funcional

3.1 Core: Chat con IA para Crear Playlists

- El usuario inicia una conversacion describiendo lo que quiere (ej: 'Quiero aprender Docker desde cero').
- La IA hace preguntas de refinamiento: nivel actual, tiempo disponible, idioma preferido, si prefiere videos cortos o cursos largos.
- La IA genera queries de busqueda optimizadas y busca videos usando web search externo (no YouTube API).
- Valida la existencia de los videos via YouTube oEmbed (gratuito, sin quota).
- Presenta opciones organizadas al usuario: videos individuales y playlists de terceros relevantes.
- El usuario selecciona los que quiere, la IA arma la playlist con un orden logico.
- La playlist se guarda en la base de datos de la app.

3.2 Player y Experiencia de Reproduccion

- Reproduccion de videos usando YouTube iframe embed (gratuito, sin API).
- Reproduccion secuencial dentro de la playlist.

- Marcar video como visto / en progreso / pendiente.
- Notas personales por video.
- Progreso general de la playlist (porcentaje completado).

3.3 Gestion de Playlists

- CRUD completo de playlists.
- Reordenar videos dentro de una playlist (drag & drop).
- Duplicar playlists.
- Tags y categorias personalizadas.
- Busqueda y filtrado de playlists propias.

3.4 Sugerencias Inteligentes en Background

- Cuando el usuario vuelve a la app, recibe sugerencias de videos para agregar a sus playlists existentes.
- Las sugerencias se generan en base al contenido actual de las playlists y el historial del usuario.
- Se ejecutan con datos cacheados y web search, sin consumir YouTube API.
- El usuario puede aceptar o descartar cada sugerencia.

3.5 Features Adicionales (Post-MVP)

- Playlists colaborativas: compartir y editar entre varios usuarios.
- Export a YouTube: sincronizar una playlist de la app con una playlist real en la cuenta de YouTube del usuario (unica funcionalidad que requiere YouTube API + OAuth).
- Playlists cross-plataforma: incluir videos de Twitch, Vimeo u otros.
- Estadisticas de uso: tiempo total visto, temas mas frecuentes, racha de estudio.
- Modo offline: descargar metadata para navegar playlists sin conexion.

---

4. Arquitectura Tecnica

4.1 Stack

Frontend: React Native (Expo) para iOS y Android.

Backend: NestJS con GraphQL (Apollo Server).

Base de datos: PostgreSQL como base principal + pgvector para busqueda semantica.

Cache: Redis para cache de resultados de busqueda y sesiones.

IA: API de Anthropic (Claude) o OpenAI para el chat conversacional y generacion de queries.

Web Search: Serper.dev o SerpAPI para buscar videos sin usar YouTube API.

Containerizacion: Docker + Docker Compose para todos los servicios.

4.2 Flujo de Datos Principal

Usuario envia mensaje en el chat > NestJS recibe via GraphQL mutation > Se envia a la API de IA con el contexto de la conversacion > La IA decide si necesita buscar videos o hacer mas preguntas > Si busca: genera queries, llama a web search, filtra resultados de YouTube, valida con oEmbed > Retorna opciones al usuario via GraphQL subscription > Usuario selecciona videos > Se crea la playlist en PostgreSQL > Se notifica al usuario via subscription.

4.3 Servicios Docker

api: NestJS + GraphQL (servicio principal).

postgres: PostgreSQL + pgvector.

redis: Cache de busquedas y rate limiting.

worker: Procesamiento async de sugerencias en background.

nginx: Reverse proxy (produccion).

4.4 Modelo de Datos (GraphQL Schema Conceptual)

User: id, email, name, preferences, createdAt.

Playlist: id, userId, title, description, tags, progress, createdAt, updatedAt.

PlaylistItem: id, playlistId, videoId, position, status (pending/watching/completed), notes, addedAt.

Video: id, youtubeId, title, thumbnail, duration, channel, cachedAt (metadata cacheada).

ChatSession: id, userId, playlistId (opcional), messages, createdAt.

Suggestion: id, userId, playlistId, videoId, reason, status (pending/accepted/dismissed), createdAt.

---

5. Dependencias Externas y Limites

5.1 YouTube

Embed (iframe): Gratuito, sin limites, sin autenticacion. Es el core del player.

oEmbed: Gratuito, sin API key. Para validar existencia y obtener metadata basica (titulo, thumbnail).

Data API v3: Solo necesaria para la feature opcional de export a YouTube. Quota de 10.000 unidades/dia. Requiere OAuth 2.0 y verificacion de Google.

La arquitectura esta disenada para que el 100% del core funcione sin la YouTube Data API.

5.2 Web Search

Serper.dev: 2.500 busquedas gratis/mes. Plan pago: $50/mes por 50k busquedas.

Alternativas: SerpAPI (100 gratis/mes), Bing Web Search (1.000 gratis/mes).

Con cache agresivo, las busquedas reales se reducen significativamente. Si 5 usuarios piden 'Docker para beginners' en la misma semana, se sirve el resultado cacheado.

5.3 IA

Anthropic Claude API: Input y output por tokens. Costo estimado por conversacion de creacion de playlist: ~$0.01-0.05 USD.

Alternativa: OpenAI GPT-4o-mini para reducir costos en sugerencias background.

---

6. Plan de Desarrollo por Fases

Fase 1: Setup e Infraestructura (Semana 1-2)

1. **Docker Compose** con servicios: api (NestJS), postgres, redis.
2. **Proyecto NestJS** con GraphQL (code-first con Apollo).
3. **Esquema de base de datos** inicial con TypeORM/Prisma: User, Playlist, PlaylistItem, Video.
4. **Proyecto React Native** (Expo) con navegacion basica y Apollo Client.
5. **Autenticacion:** registro/login con JWT (email + password o magic link).
6. **CI basico:** linting, build check.

Fase 2: Chat con IA (Semana 3-4)

1. **Integracion con API de IA** (Anthropic/OpenAI) en el backend.
2. **Modelo ChatSession** con persistencia de mensajes.
3. **System prompt** que define el comportamiento de la IA: hacer preguntas de contexto, generar queries de busqueda, presentar resultados.
4. **GraphQL mutations:** sendMessage, createChatSession.
5. **GraphQL subscriptions:** para streaming de respuestas de la IA.
6. **UI del chat** en React Native con mensajes en tiempo real.

Fase 3: Busqueda y Creacion de Playlists (Semana 5-6)

1. **Integracion con Serper.dev** (o alternativa) para web search.
2. **Servicio de extraccion:** parsear URLs de YouTube de los resultados de busqueda, extraer video IDs.
3. **Validacion via oEmbed:** verificar que los videos existen y obtener metadata.
4. **Cache en Redis:** almacenar resultados de busqueda con TTL configurable.
5. **Cache en PostgreSQL:** tabla Video con metadata persistente.
6. **Flujo completo:** la IA busca, presenta opciones en el chat, el usuario selecciona, se crea la playlist.
7. **GraphQL mutations:** createPlaylist, addVideoToPlaylist, removeVideoFromPlaylist.

Fase 4: Player y Gestion de Playlists (Semana 7-8)

1. **Componente de video player** con YouTube iframe embed en React Native (react-native-webview).

2. **Reproduccion secuencial** dentro de la playlist.
3. **UI de gestion:** lista de playlists, detalle de playlist, reordenar videos.
4. **Marcar estado por video:** pendiente, en progreso, completado.
5. **Notas por video:** campo de texto libre asociado a cada PlaylistItem.
6. **Barra de progreso** por playlist.
7. **Tags y categorias** con filtrado y busqueda.

Fase 5: Sugerencias en Background (Semana 9-10)

1. **Worker service** en Docker para procesamiento asincrono.
2. **Job scheduler:** analiza playlists del usuario periodicamente.
3. **Logica de sugerencia:** usa la IA para analizar el contenido de las playlists y generar recomendaciones con datos cacheados.
4. **Tabla Suggestion** con estado (pending/accepted/dismissed).
5. **Notificaciones push** cuando hay sugerencias nuevas.
6. **UI de sugerencias:** pantalla de bienvenida con tarjetas de sugerencias, aceptar/descartar.

Fase 6: Polish y Features Secundarias (Semana 11-12)

1. **Busqueda semantica** con pgvector: buscar en tus playlists y notas por significado.
2. **Onboarding:** flujo de primera vez que guia al usuario.
3. **Configuracion de preferencias:** idioma, duracion preferida, temas de interes.
4. **Dark mode** y pulido de UI.
5. **Testing:** unit tests en backend, integracion de GraphQL resolvers.
6. **Deploy:** backend en AWS/Railway/Fly.io, app en TestFlight/Play Store internal testing.

---

7. Estrategia de Monetizacion

7.1 Modelo Freemium

Free

- 3 playlists activas.
- 5 creaciones de playlist por IA al mes.
- Sugerencias basicas.

Pro ($4.99-7.99/mes)

- Playlists ilimitadas.
- Creaciones por IA ilimitadas.
- Sugerencias avanzadas y personalizadas.
- Export a YouTube.
- Busqueda semantica en notas.

- Playlists colaborativas.

7.2 Estructura de Costos por Usuario Activo

IA: ~$0.05 por sesion de creacion de playlist.

Web Search: ~$0.001 por busqueda (con cache, ~2-3 busquedas reales por sesion).

Infra: ~$0.01/dia por usuario activo (amortizado).

El costo marginal por usuario Pro es bajo (~$2-3/mes), dejando margen saludable con la suscripcion.

---

8. Riesgos y Mitigaciones

1. **YouTube cambia politica de embeds.** Mitigacion: es poco probable (llevan 15+ anos con iframe embed publico), pero si pasa, se puede migrar a mostrar links directos con thumbnails.
2. **Resultados de web search de baja calidad.** Mitigacion: la IA filtra y rankea, el usuario siempre tiene la decision final. Se puede complementar con scraping directo de YouTube search results.
3. **Costos de IA escalan.** Mitigacion: cache agresivo de conversaciones similares, usar modelos mas baratos para sugerencias, rate limiting en plan free.
4. **oEmbed deja de funcionar o limita.** Mitigacion: fallback a scraping de metadata desde la pagina del video, o usar YouTube Data API solo para metadata (1-3 unidades de quota, muy bajo costo).
5. **Baja retencion si las recomendaciones de la IA no son buenas.** Mitigacion: iterar el system prompt, feedback loop del usuario (thumbs up/down en sugerencias), mejorar queries con datos de uso.

---

9. Metricas de Exito

MVP (3 meses)

- App funcional en TestFlight/Play Store beta.
- Flujo completo: chat > busqueda > playlist > reproduccion.
- 10+ usuarios beta activos.

Crecimiento (6 meses)

- 100+ usuarios registrados.
- Retention D7 > 30%.
- 3+ playlists creadas por usuario promedio.
- Feature de sugerencias en background funcionando.

Monetizacion (9-12 meses)

- Lanzamiento publico en stores.
- Primeras suscripciones Pro.
- Conversion rate free > pro > 5%.

---

10. Aprendizajes Objetivo

Este proyecto esta disenado para cubrir las tecnologias objetivo del side project:

GraphQL: Schema design completo, queries, mutations, subscriptions, Apollo Server + Client, DataLoader para N+1, cache policies.

Docker: Multi-service compose, networking entre containers, volumes para persistencia, health checks, environment management.

IA: Integracion con APIs de LLM, prompt engineering, streaming de respuestas, function calling, embeddings con pgvector.

React Native: Navegacion, estado global, real-time con subscriptions, WebView para embeds, push notifications.

Arquitectura: Event-driven con workers, cache strategies, background jobs, rate limiting.