1. Arquitectura General

- Separación clara entre Frontend y Backend para facilitar el mantenimiento, escalabilidad y futuras integraciones móviles.
- Back-end desacoplado mediante GraphQL, lo que permite que tanto la web como la app móvil consuman los mismos servicios.
- Microservicios para los módulos críticos o de IA, permitiendo escalar y actualizar componentes de forma independiente.

2. Back-end

- Lenguaje y Framework:
    - Node.js con NestJS (TypeScript): rápido, moderno y con gran comunidad para APIs y microservicios.

Compatibilidad de GraphQL con NestJS

Soporte oficial y modularidad

- NestJS provee un módulo oficial llamado @nestjs/graphql que es ampliamente utilizado y documentado.
- Este módulo permite definir schemas GraphQL, resolvers, tipos y otras configuraciones utilizando conceptos familiares de NestJS como decoradores, inyección de dependencias y modularidad.
- Puedes usar tanto code first (definir tipos con TypeScript y generar schema) o schema first (definir schema GraphQL y escribir resolvers).

Integración con herramientas populares

- Soporta integración directa con apollo-server, que es el servidor GraphQL más usado y maduro en Node.js.
- Permite soporte avanzado para subscriptions (real-time updates vía WebSockets), útil para notificaciones y viralización en tu app.
- Facilita la conexión con ORMs y ODMs como TypeORM o Mongoose (usado en tu base MongoDB) para resolver datos fácilmente desde resolvers.

Beneficios para tu aplicación

- Aprovecha la arquitectura modular de NestJS para distribuir funcionalidad API entre módulos (usuarios, campañas, IA, viralización) con sus respectivos schemas y resolvers.

- La definición tipada con TypeScript facilita reutilizar tipos y validar datos de entrada/salida, mejorando calidad y mantenimiento.
- La flexibilidad de GraphQL se adapta perfectamente a la necesidad dinámica de tu frontend React para pedir sólo los datos relevantes para cada vista y evitar over-fetching.
- Permite evolución rápida del API sin la rigidez de versiones, muy útil para iterar y desplegar funcionalidades embebidas con inteligencia artificial.

Consideraciones técnicas importantes

- Requiere manejo de caché avanzado (Apollo Client ofrece capacidades para optimización cliente).
- Posibilidad de resolver consultas complejas requiere atención a performance, usando técnicas como *dataloaders* para evitar queries N+1 con la base MongoDB.
- Documentar y testear resolvers es fundamental para mantener calidad.
- Patrón de arquitectura: MVC (Model-View-Controller) para proyectos iniciales; migrar a microservicios a medida que escales.
- API Gateway para centralizar autenticación, autorización y gestión de tráfico entre frontend, backend y servicios de IA.

3. Front-end Web

- Frameworks:
    - React.js: modular, escalable y ampliamente usado para aplicaciones web ricas en interacción..
- Consumo de APIs: El frontend debe comunicarse exclusivamente a través de las APIs del backend para desacoplar la lógica de negocio.
- Preparación para mobile: Si planeas una app móvil nativa, este enfoque facilita la reutilización de lógica y servicios.

4. Base de Datos

- NoSQL: MongoDB para flexibilidad y escalabilidad horizontal, útil si manejas datos no estructurados (por ejemplo, logs, interacciones, archivos multimedia).

5. Inteligencia Artificial

- Por ahora, se van a tomar servicios externos que se puedan consumir como APIs desde NestJS

OpenAI GPT (ChatGPT, GPT-4)

- Ventajas:
    - Gran capacidad para generación de texto de alta calidad (briefs, ideas).
    - APIs flexibles y bien documentadas, compatibles con GraphQL wrappers o consumo REST desde NestJS.
    - Amplio ecosistema y comunidad.
- Enfoque: Ideal para generación automática de briefs creativos y análisis de texto descriptivo.
- Consideraciones: Costo variable según consumo; manejo fácil de prompts.

Roadmap detallado para implementación IA (Fases)

Fase 1: Preparación e investigación

- Definir datos requeridos (perfil, métricas, historial, intereses) para matching y briefs.
- Analizar formatos con GraphQL para consulta/exposición de datos IA.
- Explorar APIs OpenAI enfocadas a casos de uso.
- Definir esquema de integración con NestJS (servicios y módulos IA).
- Preparar dataset inicial anonimizado para desarrollos y pruebas.

Fase 2: Prototipado rápido

- Integrar API OpenAI GPT para generación de briefs básicos, con pruebas y ajustes de prompts.
- Construir microservicio/módulo IA NestJS para consumir API GPT.
- Implementar primer algoritmo simple de matching (reglas estáticas + puntuación).
- Crear esquema GraphQL para consulta de datos IA y generación en frontend React.
- Validar prototipo con usuarios beta y obtener feedback.

Fase 3: Optimización y escalado

- Desarrollar el modelo de matching semántico más avanzado (Vector Search, embeddings)AWS SageMaker.
- Entrenar y ajustar modelos sobre datos reales y feedback, mejorando precisión.
- Mejorar prompts y workflows de briefs con OpenAI, integrando respuestas más contextuales y creativas.
- Añadir caché y manejo de errores en servicios IA.
- Integrar analytics para medir desempeño y satisfacción.

Fase 4: Integración avanzada y automatización

- Añadir pipeline asíncrono y colas para procesar peticiones IA pesadas o en volumen.
- Implementar tests end-to-end y automatización para calidad IA y API.
- Desarrollar funcionalidades colaborativas IA: co-creación de briefs, recomendaciones en tiempo real.
- Extender uso de IA para validación antifraude y análisis de contenido (opcional).
- Escalar infraestructura (contenerización, balanceo).

Fase 5: Expansión y mejora continua

- Abrir API IA para consumo móvil y nuevos canales.
- Continuar monitoreo, ajuste de modelos y actualización según nuevo feedback y datos.
- Experimentar con nuevas funcionalidades IA (predicción de campañas exitosas, modelos generativos para contenido).
- Publicar documentación y casos de éxito.

6. Escalabilidad y Seguridad

- Autenticación y autorización: JWT (JSON Web Tokens) o OAuth2 para gestionar accesos de usuarios y marcas.
- Almacenamiento de archivos: Usa servicios de almacenamiento en la nube (AWS S3) para gestionar imágenes y videos UGC.

7. Futuro desarrollo móvil

- API First: Diseña APIs pensando en que serán consumidas tanto por la web como por apps móviles.

- Patrones recomendados para mobile: MVVM (Model-View-ViewModel) para Android/iOS, Clean Architecture si buscas máxima escalabilidad y mantenibilidad.

Resumen visual de la arquitectura

Componente	**Tecnología sugerida**	**Observaciones principales**
Frontend Web	React.js	Modular, fácil de escalar
Backend API	Node.js (NestJS)	MVC, microservicios, API Gateway
Base de datos	MongoDB	NoSQL según necesidades
IA externa	Servicios API (OpenAI, etc.)	Procesamiento de IA: matching inteligente, generación de texto, análisis de contenido, etc., consumidos vía HTTP desde NestJS.
Almacenamiento	AWS S3 / Google Cloud Storage	Para archivos multimedia
Seguridad	JWT / OAuth2	Autenticación y autorización

Recomendación final: Arranca con una arquitectura modular y desacoplada, priorizando la integración de IA vía APIs y una API robusta.

Flujo de datos y comunicación

1. El usuario interactúa con el front-end (React), que envía solicitudes HTTP (REST o GraphQL) al back-end (NestJS).
2. NestJS recibe la solicitud y, según la lógica de negocio:
    - Consulta o actualiza datos en MongoDB usando Mongoose (ODM recomendado para NestJS).
    - Si la funcionalidad requiere IA (por ejemplo, análisis de contenido o generación de recomendaciones), NestJS realiza una petición HTTP al servicio externo de IA y procesa la respuesta.
3. NestJS responde al front-end con los datos procesados o el resultado de la acción.
4. MongoDB almacena toda la información estructurada y no estructurada relevante para el negocio.

Detalles técnicos y buenas prácticas

- NestJS + MongoDB: Usa el módulo @nestjs/mongoose para definir esquemas y modelos, y manejar la persistencia de datos de forma tipada y eficiente.
- Consumo de IA externa: Implementa servicios en NestJS que utilicen librerías como axios o node-fetch para consumir APIs de IA externas. Centraliza la lógica de integración en módulos o servicios dedicados para facilitar el mantenimiento y la escalabilidad.
- Estructura modular en NestJS: Organiza el back-end en módulos (users, brands, campaigns, analytics, ai-integration, etc.) para mantener el código limpio y escalable.
- Autenticación y autorización: Implementa JWT u OAuth2 en NestJS para proteger las rutas y recursos sensibles.
- Despliegue y escalabilidad: Considera contenerizar cada componente (por ejemplo, usando Docker) para facilitar el despliegue y la escalabilidad horizontal.

- Preparación para mobile: Al tener una API desacoplada, podrás reutilizar todo el back-end y la lógica de IA cuando desarrolles la app móvil.