NADA QUE HACER.

Concierge Conversacional de Actividades para el GAM

Documentación técnica y de producto

Costa Rica · 2026

nadaquehacer.vercel.app

Proyecto	Nada Que Hacer
Stack principal	Next.js 14 · Supabase (pgvector) · Gemini API
Estado	Desplegado en producción
Fuentes de datos	5 fuentes · 360+ actividades indexadas, refresco semanal automático
Fecha	Junio 2026

01

Resumen Ejecutivo

Nada Que Hacer es un concierge conversacional de actividades para el Área Metropolitana de Costa Rica (GAM). La aplicación resuelve un problema real: la información sobre qué hacer en San José existe, pero está dispersa en múltiples sitios sin una capa de descubrimiento unificada ni una interfaz de lenguaje natural.

El sistema implementa un pipeline de IA en tres etapas: extracción estructurada de contenido web no estructurado (ingesta), búsqueda semántica por similitud vectorial (recuperación), y un concierge conversacional que refina recomendaciones en múltiples turnos de diálogo. El resultado es un producto funcional, desplegado en producción, que demuestra el uso de IA generativa más allá de la generación de texto simple.

Semanal Refresco automático de contenido	**5** fuentes de datos	**3** etapas de IA	**768** dimensiones por vector

Introducción y Planteamiento del Problema

"No hay nada que hacer en San José" es una queja recurrente entre los residentes del GAM. La paradoja es que la información sobre actividades, eventos culturales, senderos, conciertos y vida nocturna existe en abundancia, pero está fragmentada en múltiples sitios especializados sin comunicación entre sí.

El problema no es la ausencia de contenido sino la ausencia de una capa de descubrimiento inteligente. Un usuario que quiera saber qué hacer este fin de semana debe revisar manualmente Chepetown para eventos culturales, GAM Cultural para conciertos, Qué Buen Lugar para senderos, StarTicket para vida nocturna, y eticket para deportes. Ninguno de estos sitios permite preguntar en lenguaje natural, ninguno combina las fuentes, y ninguno recuerda lo que ya se mostró para refinar la búsqueda.

Objetivo del proyecto

Construir un concierge conversacional que: (1) agregue contenido de múltiples fuentes mediante scraping automatizado, (2) estructure ese contenido usando IA generativa, (3) lo indexe semánticamente para recuperación eficiente, y (4) exponga una interfaz de lenguaje natural que permita descubrir y refinar actividades de manera conversacional.

Diferenciación respecto a soluciones existentes

La distinción central no es técnica sino de diseño: Nada Que Hacer no es un motor de búsqueda con procesamiento de lenguaje natural añadido. Es un concierge. Un motor de búsqueda toma una consulta y devuelve resultados. Un concierge mantiene contexto entre turnos, hace preguntas aclaratorias cuando el intent es incompleto, y refina basándose en las reacciones del usuario. "Algo más tranquilo" o "¿puedo ir con niños?" solo tiene significado porque el sistema recuerda lo que ya sugirió.

03

Arquitectura del Sistema

Stack tecnológico

Capa	Tecnología	Justificación
Framework	Next.js 14 (App Router)	Frontend + backend en un proyecto; API routes server-side
Lenguaje	JavaScript	Menor fricción para prototipado ágil
Base de datos	Supabase (PostgreSQL + pgvector)	Gratuito, soporte nativo de vectores, experiencia previa
IA — ingesta	Gemini 2.5 Flash	Alta velocidad, bajo costo para extracción estructurada
IA — embeddings	gemini-embedding-001 (768d)	Mismo ecosistema Google, truncación Matryoshka
IA — conversación	Gemini 2.5 Pro	Mayor capacidad de razonamiento para el concierge
Despliegue	Vercel + Supabase	Tier gratuito suficiente para demo y uso académico

Nota sobre la selección de modelo de IA

Se evaluaron tres opciones principales: Claude (Anthropic), GPT-4o (OpenAI) y Gemini (Google). Gemini fue seleccionado por tres razones prácticas: (1) la autora ya contaba con una clave API validada en un proyecto RAG previo, (2) el costo de Gemini Flash para el pipeline de ingesta es significativamente menor que Claude Sonnet o GPT-4o, y (3) el rendimiento en español es comparable entre los tres modelos para este tipo de tarea. Es relevante notar que Claude fue la recomendación inicial por defecto — en retrospectiva, no era la elección óptima para este caso de uso específico.

Pipeline de Inteligencia Artificial

El diseño central del proyecto es un pipeline de IA en tres etapas. Esto lo distingue de implementaciones más simples donde la IA solo interviene en la capa de consulta. Aquí, la IA realiza trabajo real en ingesta, recuperación y conversación — tres estrategias de prompting distintas con dos modelos diferentes.

SCRAPING	INGESTA (Flash)	EMBEDDINGS	CONVERSACIÓN (Pro)
5 fuentes Cheerio / API	HTML crudo → objeto estructurado	texto → vector 768 dimensiones	top-20 + historial → JSON response

Etapa 1: Ingesta Estructurada

Cuando se procesa contenido nuevo, Gemini Flash recibe el texto y HTML crudos de un artículo o evento y extrae un objeto de actividad estructurado. Este proceso normaliza formatos inconsistentes entre fuentes, infiere campos faltantes (precio, dificultad, zona geográfica), categoriza en un vocabulario controlado de once categorías, y genera descripciones limpias en español.

El prompt de ingesta incluye reglas específicas derivadas de hallazgos durante el desarrollo: los festivales o series de eventos colapsan en una sola actividad (no una por ítem), y los tickets de servicio o transporte (ej. Fast Track aeropuerto) devuelven un array vacío. Estas reglas surgieron de pruebas contra datos reales, no de diseño a priori.

Etapa 2: Indexación Semántica con pgvector

Cada actividad extraída se convierte en un vector de 768 dimensiones usando el modelo gemini-embedding-001 con truncación Matryoshka. El texto embebido concatena título, descripción, categorías, zona y ciudad para maximizar la riqueza semántica del vector.

La ventaja sobre el enfoque de "pasar todo el contexto" es semántica, no solo de eficiencia. Una consulta como "algo tranquilo para relajarme" puede no contener palabras clave que coincidan con "jazz en un café", pero semánticamente son similares. La búsqueda por similitud coseno en pgvector detecta esa proximidad; la coincidencia de palabras clave no. Este patrón — recuperación vectorial seguida de reranking por LLM — es una arquitectura RAG

reconocida en producción.

Etapa 3: Conversación y Refinamiento

En cada turno de conversación, el sistema: (1) embebe el último mensaje del usuario, (2) recupera las 20 actividades semánticamente más cercanas via RPC de Supabase, (3) construye un bloque de contexto con las actividades, la fecha actual y condiciones climáticas estacionales, y (4) invoca a Gemini Pro con el historial completo de conversación.

Gemini Pro devuelve siempre JSON estructurado con tres campos: message (respuesta conversacional), activity_ids (IDs a renderizar como tarjetas), y awaiting_clarification (booleano que indica si el sistema está esperando más información antes de mostrar resultados). El frontend parsea este JSON y renderiza las tarjetas de forma independiente al texto conversacional.

Fuentes de Datos y Estrategia de Scraping

Fuente	Tipo	Método	Actividades
Chepetown	Blog cultural/lifestyle	Cheerio + og:image	22
GAM Cultural	Plataforma de eventos	Cheerio + JSON-LD	28
Qué Buen Lugar	Guías editoriales	API Directus + Cheerio	200
StarTicket	Ticketera (vida nocturna)	Cheerio + JSON-LD + offers[]	49
eticket.cr	Ticketera (deportes)	Cheerio + JSON-LD (59 eventos/request)	46
		Total	**360+**

Decisiones técnicas por fuente

Cada fuente presentó desafíos distintos que requirieron estrategias de extracción diferentes. Chepetown y GAM Cultural son sitios con HTML renderizado en servidor, donde Cheerio extrae contenido directamente. Qué Buen Lugar es una aplicación Next.js respaldada por Directus, con páginas de listado renderizadas en cliente — fue necesario usar la API REST pública de Directus para enumeración y Cheerio en las páginas de detalle para el contenido completo.

StarTicket presenta Cloudflare como capa de protección, pero un User-Agent de navegador es suficiente para acceder sin restricciones. El esquema JSON-LD de Event en estas páginas incluye un array offers[] con disponibilidad por tier (InStock, LimitedAvailability, SoldOut), lo que permite filtrar eventos agotados antes de la ingesta. eticket.cr usa ASP.NET clásico y expone todos sus eventos en JSON-LD en una sola request a /eventos.aspx, siendo el scraper más eficiente.

Consideraciones éticas

El uso de scraping plantea preguntas legítimas sobre los derechos del contenido original. La postura adoptada en este proyecto: cada resultado muestra atribución explícita a la fuente (nombre del sitio, enlace directo al artículo original), lo que difiere del scraping extractivo sin crédito. OpenTable fue explícitamente excluido por contar con una política robots.txt que prohíbe el acceso automatizado y mecanismos activos de detección — técnicamente eludibles

pero legalmente problemáticos.

06

Diseño del Sistema de Prompts

El sistema utiliza dos prompts cuidadosamente diseñados. El primero controla la extracción estructurada durante la ingesta; el segundo define el comportamiento del concierge conversacional. Ambos reflejan decisiones deliberadas que surgieron de iteraciones contra datos reales.

Prompt de ingesta: principios de diseño

El prompt de ingesta instruye a Gemini Flash a devolver siempre un array JSON válido — sin preámbulo, sin markdown. Las reglas críticas surgieron de hallazgos durante el desarrollo:

- Los festivales o series de eventos colapsan en una actividad, no una por ítem
- Si el content es un ticket de servicio o transporte (no una actividad), retornar []
- El campo "gratuito" en categories se añade automáticamente si price es gratis
- difficulty se popula únicamente para actividades al aire libre o senderismo

La segunda regla surgió después de que eticket.cr ingresara tickets de "Fast Track SJO" (cola rápida del aeropuerto) como actividades. Tres filas fueron eliminadas manualmente de Supabase y la regla fue añadida al prompt para prevenir reingresos.

System prompt: diseño del concierge

El system prompt del concierge fue diseñado con cuatro principios rectores:

- **Resultados primero:** Si el usuario provee cualquier señal — zona, vibe, categoría, día — mostrar resultados inmediatamente. No retener.
- **Una sola pregunta aclaratoria:** Solo preguntar antes de mostrar resultados si el query es genuinamente ambiguo ("quiero hacer algo" sin más contexto).
- **Lenguaje espejo:** Responder en el idioma en que escribe el usuario. Español por defecto.
- **Tono calibrado:** Útil, casual y cálido — pero sin fingir ser humano. Sin jerga forzada, sin exclamaciones performativas.

El formato de output es JSON estructurado con tres campos: message, activity_ids (array de UUIDs), y awaiting_clarification (booleano). Esto separa la capa conversacional de la capa de renderizado de tarjetas en el frontend — una decisión arquitectónica que permite que ambas

evolucionen independientemente.

Conciencia climática y estacional

Costa Rica tiene microclimas muy distintos. La temporada lluviosa (mayo–noviembre) no aplica igual en todo el país: el Valle Central suele tener mañanas despejadas con lluvias por la tarde, mientras que zonas altas como Bajos del Toro, Poás o Chirripó pueden tener senderos cerrados por derrumbes en septiembre y octubre.

Se consideró integrar una API de clima en tiempo real (Open-Meteo, gratuita, sin clave). Esta opción fue descartada por una razón de producto: la mayoría de las consultas son de planificación futura ("este fin de semana", "en octubre") y no de condiciones en tiempo real. El conocimiento estacional regional — codificado como una sección CLIMA Y TEMPORADA en el system prompt — es más útil que una lectura de precipitación del momento actual.

07

Tracking de Disponibilidad de Eventos

Las fuentes de ticketera (StarTicket y eticket.cr) exponen un array offers[] en su esquema JSON-LD Event. Cada oferta incluye un campo availability con uno de tres valores: InStock, LimitedAvailability, o SoldOut. Este dato es relevante para el producto: un concierge que recomienda tres conciertos agotados no es útil.

El sistema deriva un campo availability consolidado por actividad: sold_out si todos los tiers están agotados, limited si alguno tiene LimitedAvailability pero ninguno tiene InStock, y available en cualquier otro caso. Los eventos sold_out son filtrados del vector search antes de pasarse a Gemini Pro. Las tarjetas con status limited muestran un badge "Pocas entradas".

Las fuentes no-ticketera (Chepetown, GAM Cultural, Qué Buen Lugar) tienen availability nulo, interpretado como "disponible o no aplica".

Interfaz de Usuario

La interfaz implementa dos estados distintos con navegación entre ellos. Esta separación refleja una decisión de producto: la página de inicio es una vitrina de descubrimiento pasivo; la vista de chat es un espacio de descubrimiento activo y conversacional.

Estado 1: Homepage

Encabezado en verde oscuro con el logotipo, tagline en naranja, y una barra de búsqueda prominente. El encabezado incluye una capa de iconos de actividades (música, montaña, cámara, hoja, café) en muy baja opacidad como textura decorativa. El cuerpo en crema contiene filtros de categoría como pills con icono, seguidos de tres secciones curadas:

- **Destacados esta semana:** Mix de próximos eventos con imágenes
- **Esta noche:** Actividades con fecha igual a hoy, principalmente de fuentes ticketera
- **Al aire libre:** Actividades de senderismo y exterior, filtradas estacionalmente

Estado 2: Chat

Al enviar una consulta desde la homepage, la URL navega a /chat llevando el mensaje como primer turno. La interfaz muestra burbujas de chat (mensajes de usuario a la derecha, respuestas del concierge a la izquierda) con tarjetas de actividad renderizadas inline debajo de cada respuesta. Cada tarjeta incluye imagen (con placeholder gris si no hay imagen disponible), título en blanco sobre overlay oscuro, categoría en naranja, zona, precio, y un enlace "Ver más" a la fuente original.

Paleta y decisiones de diseño

Elemento	Valor	Uso
Verde oscuro	#1B3D2F	Encabezado, elementos primarios, pills activos
Naranja	#E8401C	Acento, CTA, tags de categoría, tagline
Crema	#F5F0E8	Fondo del cuerpo
Blanco	#FFFFFF	Tarjetas, pills inactivos, barra de búsqueda

09

Resultados y Estado del Proyecto

El proyecto está desplegado en producción en nadaquehacer.vercel.app. El pipeline completo funciona de extremo a extremo: scraping → extracción estructurada → embeddings → búsqueda semántica → concierge conversacional.

Validaciones realizadas

- Pipeline de ingesta validado contra 2 artículos por fuente antes del scrape completo
- 250 actividades iniciales scrapeadas e indexadas en ~11 minutos (primera corrida)
- Deduplicación verificada: re-run sobre 266 filas existentes produce 0 inserciones
- Búsqueda semántica verificada: consulta "senderismo cerca de Turrialba" retorna actividades de senderismo — no actividades aleatorias
- Conciencia estacional verificada: el sistema advierte sobre zonas de riesgo en temporada lluviosa sin sobre-filtrar actividades en el Valle Central
- Tracking de disponibilidad verificado: CIRCUITO AGUAS ABIERTAS (sold_out) correctamente excluido de recomendaciones
- Refresco automático implementado: GitHub Actions cron ejecutado exitosamente, 15 nuevas actividades añadidas en primera corrida (5m 40s), todos los scrapers alcanzados

Comportamiento del concierge

El sistema maneja correctamente los escenarios centrales del diseño: consultas específicas ("quiero hacer senderismo en Turrialba este finde") devuelven resultados directos con advertencia estacional si aplica; consultas vagas ("quiero hacer algo") generan una pregunta aclaratoria; refinamientos conversacionales ("algo más barato", "¿para niños?") producen nuevas recomendaciones sin repetir las ya mostradas. El sistema responde correctamente tanto en español como en inglés según el idioma del usuario.

Limitación identificada: consultas de agregación

Una consulta como "¿cuál fin de semana tiene más opciones de vida nocturna de aquí a diciembre?" requiere una agregación SQL sobre el dataset completo — no búsqueda vectorial, que solo ve los 20 documentos semánticamente más cercanos. El sistema maneja esto con un

redirect graceful: informa brevemente que no puede ver el calendario completo y sugiere preguntar por tipo de evento o fecha específica. La solución técnica correcta sería un tercer código path que detecta intent de agregación y lo redirige a una consulta SQL — identificado como trabajo futuro.

Limitaciones Conocidas y Trabajo Futuro

Limitaciones actuales

Limitación	Causa	Solución propuesta
Refresco automático de scraping	GitHub Actions cron implementado: script semanal ejecutado en runner de GH, sin límite de tiempo, escribe dir	Resuelto ✓
Sin datos de restaurantes	OpenTable bloquea scraping (robots.txt + TLS); Google Places API $200/mes crédito gratuito, suficiente para den	Integrar Places API
Dedup a nivel de artículo	Si un artículo upstream se actualiza no se re-ingesta	Re-ingesta basada en scraped_at (TTL configurable)
Consultas de agregación limitadas	Vector search retorna top-20, no agrega sobre todos los datos	Tercer código path con detección de intent de agregación → SQL

Líneas de desarrollo futuro

- **Enriquecimiento en ingesta:** Pedir a Gemini Flash que evalúe cada actividad durante la extracción, añadiendo campos como vibe (relajado/romántico/aventurero), best_for (pareja/familia/grupo), uniqueness (1-5), y indoor_outdoor. Esto haría los embeddings más ricos y las recomendaciones más precisas.
- **Restaurantes vía Google Places API:** La API provee nombre, dirección, rating, horarios, nivel de precio y fotos. Los horarios son particularmente valiosos para el concierge ("cena romántica el sábado que no cierre temprano").
- **eticket Queue-it:** Algunos eventos de alta demanda en eticket están detrás de Queue-it. node-fetch alcanza el límite de 20 redirecciones y omite esos eventos. Fix: aumentar el límite o detectar la URL de cola y descartar el token.
- **API de clima como complemento:** Open-Meteo (gratuito, sin clave) podría añadir condiciones en tiempo real para consultas del tipo "qué puedo hacer hoy" sin reemplazar el conocimiento estacional.

Conclusiones

Nada Que Hacer demuestra que la IA generativa puede integrarse de manera significativa en múltiples etapas de un producto de información — no solo en la interfaz de usuario. El pipeline de tres etapas (extracción estructurada, indexación semántica, concierge conversacional) produce un sistema cualitativamente diferente a un buscador con procesamiento de lenguaje natural: uno que razona sobre contexto acumulado, detecta cuándo necesita más información, y refina iterativamente en lugar de responder en una sola pasada.

Las decisiones más interesantes del proyecto no fueron las técnicas sino las de producto. La distinción entre "concierge" y "motor de búsqueda" definió toda la arquitectura de estado de la conversación. La decisión de no usar la API de clima en tiempo real — favorable a conocimiento estacional más robusto — refleja que la claridad sobre el caso de uso real del usuario (planificación futura, no información inmediata) importa más que añadir capacidades técnicas.

El proyecto también ilustra el valor de construir de forma incremental y validar contra datos reales: la regla de "tickets de servicio devuelven []", el colapso de festivales en una sola actividad, el enfoque híbrido API+Cheerio para Qué Buen Lugar, el campo awaiting_clarification en lugar del campo redundante clarifying_question — todas estas decisiones surgieron de encontrar problemas concretos, no de diseño anticipado.

Proyecto en producción: nadaquehacer.vercel.app	**Repositorio:** github.com/SofiAnabelle/nadaquehacer