TripSense: Aplicación Web de Recomendaciones Geolocalizadas con Inteligencia Artificial

Bruno Morán Vivanco

Resum– Desarrollo de TripSense, una aplicación web que combina geolocalización con inteligencia artificial para generar itinerarios de viaje personalizados según los intereses y preferencias del usuario. El sistema permite buscar puntos de interés, organizar actividades por días y consultar recomendaciones generadas automáticamente por un modelo de lenguaje. La aplicación construida en React, Node.js, Express y MongoDB integra APIs externas como Google Maps y Gemini para enriquecer la experiencia del usuario. En las características del proyecto están el diseño modular del frontend, la arquitectura REST del backend y un motor de generación de itinerarios mediante prompts estructurados. Los resultados muestran una herramienta útil, funcional y escalable.

Paraules clau– Aplicación web, recomendaciones geolocalizadas, generación de itinerarios, inteligencia artificial, Google Maps, Gemini.

Abstract– Development of TripSense, a web application that combines geolocation with artificial intelligence to generate personalised travel itineraries according to the user's interests and preferences. The system allows users to search for points of interest, organise activities by days and consult recommendations automatically generated by a language model. The application built in React, Node.js, Express and MongoDB integrates external APIs such as Google Maps and Gemini to enrich the user experience. The project features a modular frontend design, backend REST architecture and an itinerary generation engine using structured prompts. The results show a useful, functional and scalable tool.

Keywords– Web application, geolocalised recommendations, itinerary generation, artificial intelligence, Google Maps, Gemini.

1 Introducción - Contexto del trabajo

En un mundo cada vez más conectado, planificar un viaje va mucho más allá de reservar vuelos y hoteles. La experiencia del viajero empieza mucho antes, en la búsqueda de información, la exploración de lugares de interés y la organización eficiente del tiempo disponible. No obstante, este proceso puede presentar desafíos importantes debido a la cantidad de opciones disponibles, la falta de herramientas centralizadas y la necesidad de adaptar el itinerario a los intereses particulares de cada persona.

En este contexto, surge TripSense, una aplicación web desarrollada con el objetivo de facilitar la planificación de viajes mediante el uso de geolocalización e inteligencia artificial. TripSense permite explorar puntos de interés cercanos, o del próximo destino del usuario, y organizarlos en un itinerario diario, además de ofrecer su funcionalidad principal: generar automáticamente propuestas de itinerarios adaptadas a las preferencias del usuario a través de modelos de inteligencia artificial.

El presente trabajo describe el diseño y desarrollo completo de esta aplicación, que se apoya en tecnologías modernas como React, Node.js, MongoDB, la API de Google Maps y el modelo Gemini de Google para la generación automatizada. El objetivo principal ha sido crear una solución funcional, personalizable y escalable que proporcione valor real al viajero digital.

• E-mail de contacte: bruno.moranvivanco@gmail.com
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Sergio Bachiller (Àrea de Ciències de la Computació i Intel·ligència Artificial)
• Curs 2024/25

2 OBJETIVOS

El objetivo general del presente Trabajo Final de Grado es diseñar y desarrollar una aplicación web que optimice la planificación de viajes mediante geolocalización e inteligencia artificial, proporcionando herramientas para la organización eficiente de actividades.

Para alcanzar este propósito, se han definido los siguientes objetivos específicos:

- Diseñar una arquitectura modular y escalable basada en tecnologías web modernas. Utilizando React para el frontend, Node.js con Typescript para el backend y MongoDB como base de datos principal.
- Implementar funciones de búsqueda y visualización de puntos de interés en un mapa, utilizando la API de Google Maps, filtrando por tipo de lugar y proximidad a la ubicación del usuario o destino seleccionado.
- Desarrollar un sistema de creación y edición de itinerarios, permitiendo al usuario añadir y eliminar actividades diarias, así como elegir medios de transporte entre ellas y calcular automáticamente rutas y tiempos estimados.
- Integrar un sistema de recomendación basado en inteligencia artificial que, a partir de un formulario con destino, fechas y preferencias del usuario, genere automáticamente una propuesta de itinerario estructurado en días.
- Persistir la información de los usuarios y de los itinerarios en una base de datos, permitiendo gestionar, itinerarios públicos y privados, así como habilitar funcionalidades sociales como perfiles, seguidores y comunidad.

Estos objetivos permiten abordar el problema de planificación de viajes desde una perspectiva técnica y práctica, obteniendo una mejora en el proceso de organización de un viaje.

3 ESTADO DEL ARTE

La planificación de itinerarios turísticos personalizados ha sido una necesidad recurrente en el ámbito de las aplicaciones web. En los últimos años, han proliferado herramientas que permiten al usuario consultar puntos de interés, planificar actividades y compartir experiencias de viaje. No obstante, la mayoría de estas soluciones presentan limitaciones en cuanto a personalización, automatización y adaptación a las preferencias individuales del usuario.

Plataformas como Google Travel, o TripIt permiten construir itinerarios de forma manual, integrando mapas, horarios y recomendaciones generales [1]. Sin embargo, en estos casos, la interacción suele depender exclusivamente de la acción directa del usuario, sin incorporar mecanismos inteligentes que generen propuestas automáticas o personalizadas basadas en entradas estructuradas como intereses o duración del viaje.

En paralelo, otras aplicaciones como Komoot [2] ofrecen funciones avanzadas de planificación, pero están especializadas en nichos concretos, como puede ser ciclismo y senderismo, o bien requieren suscripciones para acceder a funcionalidades completas. Algunas redes sociales deportivas como Strava [3] fomentan la participación de los usuarios a compartir en forma de publicación los registros de las actividades físicas realizadas, pero limitan la creación de planes o rutas.

Desde el punto de vista tecnológico, la aparición de APIs de geolocalización y de modelos de lenguaje con inteligencia artificial ha abierto nuevas posibilidades para automatizar la creación de contenidos y recomendaciones adaptadas. Sin embargo, su integración en productos turísticos finales aún es limitada, en parte por la complejidad técnica y por la falta de herramientas puente que las conecten de forma intuitiva para el usuario final.

En este contexto, TripSense propone una aproximación híbrida e innovadora, combinando:

- Un sistema visual e interactivo de gestión de itinerarios.
- Integración con APIs externas para obtener información geográfica detallada y actualizada.
- Un generador inteligente de itinerarios basado en inteligencia artificial, que produce propuestas completas a partir de un simple formulario.

Este enfoque permite superar las barreras observadas en soluciones previas, aportando un valor diferencial tanto en términos de experiencia de usuario como de automatización y adaptabilidad, debido a la utilización de herramientas con las que la mayoría de los usuarios están familiarizados.

4 METODOLOGÍA

Para el desarrollo del proyecto se ha seguido una metodología ágil basada en Kanban, adaptada a la naturaleza individual del proyecto [4]. La flexibilidad que ofrece Kanban ha permitido priorizar tareas de forma dinámica, adaptando el flujo de trabajo a medida que surgían imprevistos o retrasos, y organizar de forma visual el trabajo, facilitando una visión clara del progreso [5].

Se ha utilizado la herramienta Jira [6] para la gestión de las tareas categorizadas en columnas *Por hacer*, *En curso* y *Listo*. Cada tarjeta representa una funcionalidad o un componente del sistema.

A nivel técnico, la utilización de GitHub [7] como repositorio ha facilitado el seguimiento de las versiones del proyecto y la posibilidad de crear ramas para valorar diferentes opciones de implementación.

La toma de decisiones técnicas se ha realizado de forma progresiva durante el desarrollo, priorizando la funcionalidad y la estabilidad en cada uno de los componentes antes de avanzar al siguiente.

5 REQUISITOS

El desarrollo del proyecto ha estado guiado por un conjunto de requisitos funcionales y no funcionales definidos en las primeras etapas del proyecto. Estos requisitos han

servido como base para la toma de decisiones técnicas, el diseño de la arquitectura y la validación de la solución final.

5.1. Requisitos funcionales

- El sistema debe detectar la ubicación del usuario con su consentimiento.
- El usuario debe poder seleccionar manualmente una ubicación.
- El sistema debe mostrar puntos de interés en un mapa.
- El usuario debe poder crear, editar, guardar y eliminar itinerarios.
- El usuario debe poder añadir y eliminar actividades en un itinerario.
- El sistema debe calcular rutas y distancias entre actividades utilizando la API de mapas.
- El sistema debe permitir a los usuarios crear perfiles y seguir a otros usuarios.
- El usuario debe poder compartir itinerarios en forma de publicaciones.
- El sistema debe integrar un modelo de lenguaje para generar itinerarios de viaje en la localización que el usuario elija.
- Las respuestas generadas por el modelo de lenguaje debe ser coherentes, útiles y contextualizadas según la ubicación y preferencias del usuario.

5.2. Requisitos no funcionales

- El sistema debe responder a las solicitudes del usuario en menos de dos segundos en condiciones normales de uso.
- La integración del modelo de lenguaje no debe afectar significativamente el tiempo de respuesta de la aplicación.
- La aplicación debe ser capaz de manejar un aumento en el número de usuarios sin bajar el rendimiento.
- La base de datos debe ser escalable para soportar un crecimiento en el volumen de datos.
- El sistema debe garantizar la privacidad de los datos del usuario.
- Las contraseñas de los usuarios deben almacenarse de forma segura utilizando técnicas de *hashing*.
- La interfaz de usuario debe ser intuitiva y fácil de usar, con un tiempo de aprendizaje mínimo para nuevos usuarios.
- El sistema debe garantizar que esté disponible para los usuarios en todo momento.
- La aplicación debe ser compatible con los principales navegadores web.

6 Planificación

Para garantizar una implementación eficiente del proyecto, la planificación se ha estructurado en los objetivos específicos y requisitos definidos previamente. Antes de iniciar el desarrollo funcional, se crearon prototipos de diseño de la aplicación que sirvieron como punto de partida para guiar tanto la implementación de la interfaz como la experiencia de usuario, que se pueden ver en las figuras 18.

El desarrollo del sistema se ha dividido en cuatro fases principales.

6.1. Fase 1: Geolocalización y mapas

La primera fase se centró en la integración de las funciones relacionadas con la geolocalización y el uso de mapas. Esta fase se desglosa en aportar soluciones relacionadas con mecanismos para detectar la ubicación actual del usuario, visualizar destinos seleccionados y mostrar puntos de interés en el mapa.

6.2. Fase 2: Gestión de itinerarios

La siguiente fase se desarrollaron las funcionalidades que pertenecen a la parte de itinerarios de la aplicación. Durante la ejecución de esta fase se implementó la creación, edición y visualización de itinerarios, las interfaces de usuario y la lógica de cálculos como la duración del itinerario y rutas óptimas según los modos de transporte que el usuario ha elegido previamente.

6.3. Fase 3: Funcionalidades sociales

Como tercera fase, el objetivo fue incorporar características propias de una red social. Se diseñaron los perfiles de los usuarios y el funcionamiento de la red social, en la que los usuarios pueden compartir con la comunidad sus itinerarios e interactuar entre ellos.

6.4. Fase 4: Integración de inteligencia artificial

Y como última fase, es la integración de inteligencia artificial. Esta fase consiste en investigar los modelos de lenguaje adecuados para que ofrezcan asistencia en la selección de actividades y poder generar itinerarios personalizados para los usuarios.

Cada fase se planificó de forma que, al completarse cada una, sirviera de base en caso de que otra fase dependiera de otra, como por ejemplo, la fase de integración de la inteligencia artificial necesitaba que los itinerarios tuvieran un modelo de datos consistente para que las respuestas generadas aportaran valor al usuario.

7 Diagrama de casos de uso

El diagrama de la figura 19 muestra los casos de uso de la aplicación y las interacciones entre actores y funcionalidades del sistema. Este modelo UML divide las capacidades del sistema en cuatro dominios funcionales, estableciendo

las relaciones jerárquicas y de dependencia entre las diferentes acciones que pueden realizarse en la plataforma.

Los dominios funcionales son gestión de usuarios, gestión de itinerarios, exploración y comunidad. Como actores que interactúan con la aplicación, tenemos usuario no registrado, usuario registrado y sistema de IA.

8 DESARROLLO

El desarrollo del proyecto se ha llevado a cabo mediante una arquitectura modular basada en tecnologías web modernas, que permiten separar claramente la lógica de presentación, negocio y persistencia de datos.

A continuación se detallan los principales componentes desarrollados y las decisiones técnicas adoptadas.

8.1. Frontend

El frontend de TripSense ha sido desarrollado utilizando React [8], una biblioteca de JavaScript ampliamente adoptada en el desarrollo de interfaces web dinámicas. Se ha optado por una estructura basada en componentes funcionales, lo que permite una gestión del estado más modular y una lógica de componentes más limpia.

Todo el código está escrito en TypeScript [9], lo que proporciona un tipado estático que mejora la fiabilidad del desarrollo y previene errores comunes durante la ejecución. Esta decisión ha permitido definir interfaces compartidas entre frontend y backend, facilitando la validación y la coherencia de los datos a lo largo de todo el sistema.

La estructura del proyecto se ha organizado siguiendo un modelo por responsabilidades técnicas, donde los archivos se agrupan según su propósito.

Para el diseño visual se ha utilizado CSS [10] modular, permitiendo tener un control detallado sobre el aspecto de la interfaz, evitar dependencias innecesarias y seguir una estrategia de estilos personalizada y coherente con la identidad visual del proyecto.

El sistema de rutas se ha implementado con React Router [11], permitiendo la navegación entre páginas. También se ha gestionado el acceso a rutas privadas mediante una capa de control que redirige a usuarios no autenticados.

En cuanto al consumo de datos, se ha utilizado la librería Axios [12] para realizar llamadas HTTP al backend. Axios ha resultado muy potente para el proyecto, debido a que es un cliente HTTP tanto para el servidor como para la parte del cliente.

Durante el desarrollo, se ha utilizado Vite [13] como bundler por su rapidez en el arranque y recarga en caliente, junto con un entorno de desarrollo en local conectado al backend.

8.2. Backend

El backend de TripSense ha sido desarrollado utilizando Node.js [14] con el framework Express [15], ambos ampliamente utilizados en entornos de desarrollo web modernos por su rendimiento, simplicidad y ecosistema. Se ha empleado TypeScript como lenguaje principal para dotar al proyecto de un tipado fuerte, mayor seguridad en tiempo de desarrollo y una estructura más robusta frente a errores.

La organización del backend sigue el patrón de arquitectura en capas [16], dividiendo la lógica en:

- Modelos: Representan la estructura de los datos en la base de datos.
- Controladores: Contiene la lógica que interactúa con los modelos.
- Rutas: Definen los endpoints y los conectan con los controladores.
- Middlewares: Funciones que procesan las solicitudes antes de llegar a los controladores.
- Configuraciones: Gestión de la conexión a la base de datos.
- Tipos: Definiciones de interfaces de TypeScript para mantener el tipado.

Esta estructura modular favorece la escalabilidad del sistema y mejora el mantenimiento a largo plazo, permitiendo añadir nuevas funcionalidades o modificar las existentes con bajo acoplamiento entre componentes.

La base de datos utilizada es MongoDB, un sistema NoSQL orientado a documentos que permite almacenar estructuras flexibles y anidadas [17] como los itinerarios organizados por días y actividades. Se ha utilizado Mongoose [18] para definir esquemas con restricciones, tipos de datos y validadores.

Los principales modelos definidos son:

- User: Representa a los usuarios de la aplicación. Guarda la información básica como nombre, correo electrónico, contraseña. También mantiene las referencias de los usuarios seguidos y seguidores.
- Itinerary: Almacena los itinerarios de viaje creados por los usuarios. Contiene una estructura jerárquica en el que se divide el itinerario en días, y cada día tiene actividades relacionadas.
- Post: Representa las publicaciones compartidas por los usuarios en la comunidad. Permite una descripción opcional y mantiene un array de usuarios que han dado ”me gusta”.

Esta estructura permite modelar de forma natural la lógica de la aplicación, evitando la necesidad de estructuras relacionales complejas. El diagrama de la base de datos se encuentra en la figura 20.

Desde el backend se ha gestionado la integración con servicios de terceros.

En diferentes endpoints se utiliza Google Maps Platform [19] para buscar puntos de interés por ubicación, calcular rutas y obtener tiempos estimados entre actividades. Las solicitudes del frontend se envían al backend, que actúa como proxy seguro y procesa la respuesta.

En el apartado de la integración de la inteligencia artificial, el frontend envía un formulario al backend, donde se construye un prompt estructurado para que se envíe al modelo de lenguaje con la Gemini API de Google AI. La respuesta es un JSON con la estructura de datos necesaria para crear un itinerario según los días que el usuario ha solicitado.

8.2.1. Geocodificación

El sistema incluye funcionalidades de geocodificación directa e inversa mediante la API de Google Maps. Estas operaciones permiten traducir una ubicación textual, como puede ser el nombre de una ciudad, a coordenadas geográficas especificadas en latitud y longitud y viceversa, detectando desde dónde se hace la petición y obteniendo el nombre de la ciudad. Esta capacidad es esencial para vincular correctamente los itinerarios con una ubicación geográfica concreta y para filtrar los puntos de interés próximos a una ciudad o posición del usuario.

La función de geocodificación directa recibe como entrada una cadena de texto con el nombre del lugar introducido por el usuario y consulta la API de geocodificación de Google Maps [20]. A partir del resultado, se extraen las coordenadas, el nombre de la localidad y su dirección formateada.

Por otro lado, la geocodificación inversa parte de unas coordenadas geográficas y devuelve el nombre de la localidad correspondiente. El algoritmo implementado analiza el conjunto de resultados devueltos por la API, priorizando aquellos que incluyan componentes del tipo *locality* que es la ciudad, *administrative area 2* que es la provincia, o *administrative area 1* que es la región, para así ofrecer una identificación lo más precisa posible del lugar. Esta funcionalidad se utiliza especialmente cuando el usuario accede desde su ubicación actual sin introducir manualmente una ciudad en el buscador.

Ambos procesos se encapsulan en rutas del backend protegidas mediante middlewares, y los resultados son devueltos al frontend en formato JSON normalizado.

8.2.2. Puntos de interés

La funcionalidad de búsqueda de puntos de interés es una de las piezas clave del sistema. Permite al usuario descubrir lugares relevantes cercanos a una ubicación determinada con la ayuda de las funcionalidades explicadas en el apartado 8.2.1.

Para ello, se ha implementado un sistema que realiza llamadas consecutivas al endpoint *nearby search* de la API de Google Places [21] para cada una de las categorías definidas (restaurantes, hoteles, museos, atracciones, parques, cafeterías). Esta lógica está centralizada en una función que recibe las coordenadas geográficas, el identificador de categoría y una clave de API, y devuelve una lista estructurada de objetos POI (Point Of Interest). La definición de Point Of Interest se puede ver en la figura 23.

Cada punto de interés obtenido se transforma en un objeto con la siguiente estructura:

- id: identificador interno.
- name: nombre del lugar.
- description: dirección aproximada o vecindario.
- location: coordenadas geográficas representadas por longitud y latitud.
- category: categoría semántica.

El backend ofrece dos rutas para acceder a estos datos: una que obtiene los puntos de interés cercanos a las coordenadas actuales del usuario con la función *getPOIsNearby*, y la otra ruta que los obtiene a partir del nombre de una ciudad con la función *getPOIsByCity*. Ambas rutas necesitan que se haya realizado previamente la geolocalización guardando las coordenadas gracias a los middlewares.

Los resultados se agrupan y devuelven en una única respuesta optimizada, con todos los puntos de interés organizados por categoría. Esta funcionalidad permite enriquecer la experiencia de usuario en la fase de creación y edición de itinerarios, ofreciendo un catálogo contextualizado de actividades relevantes.

8.2.3. Cálculo de rutas

La siguiente funcionalidad es el cálculo automático de rutas entre actividades dentro de un itinerario. Esta funcionalidad se encarga de determinar, para cada día del viaje, las rutas óptimas entre los puntos de interés seleccionados por el usuario, teniendo en cuenta los modos de transporte disponibles.

Para ello, se utiliza el servicio Directions API de Google Maps [22], el cual permite obtener múltiples rutas entre dos ubicaciones geográficas, especificando el modo de transporte (por ejemplo, a pie o en transporte público). El sistema evalúa cada una de las opciones y selecciona la que presenta la menor duración estimada.

El proceso sigue los siguientes pasos:

1. Obtención del itinerario: se localiza el itinerario en la base de datos a partir del identificador recibido en la petición.
2. Iteración por días: para cada día del itinerario, se recorren las actividades planificadas.
3. Cálculo de rutas entre pares de actividades: se calculan todas las rutas posibles entre dos actividades consecutivas utilizando los modos de transporte definidos (por defecto: ”walkingƶ ”transit”). Se selecciona la más rápida.
4. Actualización de actividades: la información de duración, distancia, modo y ruta completa se almacena dentro de cada actividad bajo el campo *routeToNext*.
5. Resumen diario: se calcula y almacena la distancia total y duración del día, junto con los segmentos de ruta intermedios.
6. Resumen global del itinerario: se agregan los datos de cada día para generar un resumen total de duración y distancia del viaje completo, almacenado en el campo completeRoute.

Las distancias se formatean en metros o kilómetros según la magnitud, y las duraciones se expresan en minutos u horas. Este formato mejora la presentación en el frontend.

8.2.4. Integración con inteligencia artificial

La incorporación de la inteligencia artificial en este proyecto consiste en la generación automática de itinerarios turísticos según las preferencias del usuario utilizando un modelo de lenguaje, en concreto Gemini 2.0 Flash [23] proporcionado por Google AI. Esta funcionalidad permite al usuario generar itinerarios completos en función de sus

preferencias y fechas, eliminando la necesidad de planificar manualmente las actividades.

El sistema de generación se activa a partir de un formulario que recoge información clave del usuario: destino, fechas de inicio y fin del viaje, medios de transporte preferidos, intereses personales y una lista de puntos de interés obtenidos previamente de la ubicación de destino especificada.

Los puntos de interés se mezclan siguiendo el algoritmo de Fisher-Yates [24] para evitar que las opciones que recibe el modelo de lenguaje sean siempre las mismas.

A partir de estos datos, el backend construye de manera dinámica un *prompt* en lenguaje natural que se envía al modelo de Gemini. Este *prompt* incluye: una descripción del viaje (destino y duración), los medios de transporte que se deben tener en cuenta, una lista personalizada de puntos de interés disponibles en esa ciudad, y observaciones adicionales que el usuario haya introducido manualmente (figura 24).

Además del prompt en lenguaje natural, el sistema adjunta un esquema JSON estructurado que define exactamente el formato que debe tener la respuesta generada por el modelo (figuras 21 y 22). Este esquema actúa como una plantilla formal para la función *generateContent* [25] del SDK oficial de Google GenAI, lo que permite generar directamente una salida en formato JSON válido. Gracias a esta técnica, se minimizan los errores habituales en tareas de texto libre y se garantiza que los datos devueltos cumplen la estructura esperada por el sistema, incluidos los identificadores, fechas e información de los puntos de interés como si se construyese el itinerario de forma manual.

El modelo seleccionado, Gemini 2.0 Flash, ha sido elegido por su facilidad de uso, su integración directa mediante SDK oficial de Google y por ofrecer una capa gratuita suficiente para los fines del proyecto. Esta elección ha permitido incorporar inteligencia artificial generativa de forma rápida, fiable y sin necesidad de infraestructuras adicionales, aprovechando además la capacidad del modelo para estructurar contenido semántico complejo a partir de datos proporcionados por el usuario.

8.3. Estrategia de test

Para garantizar la fiabilidad funcional de la aplicación durante el desarrollo, se ha seguido una estrategia de testeo basada en dos enfoques complementarios: pruebas manuales continuas y pruebas end-to-end (E2E) al finalizar bloques de funcionalidades.

Durante la implementación de cada módulo, se realizaron pruebas manuales sistemáticas para validar visual y funcionalmente el comportamiento esperado. Estas pruebas incluían el flujo de usuario desde la creación de itinerarios, interacción con mapas, selección de puntos de interés, hasta la publicación en la comunidad. Se verificó la correcta visualización de datos, la gestión de formularios y la actualización dinámica del estado de la aplicación.

Adicionalmente, se realizaron pruebas end-to-end tras cada fase principal del proyecto, como la creación de itinerarios, edición y guardado, cálculo de rutas y uso del generador IA. Estas pruebas se realizaron desde el frontend, validando la interacción completa con el backend y el comportamiento de la API.

9 RESULTADOS

En esta sección se muestran los principales resultados obtenidos tras la implementación del proyecto. Se muestra el funcionamiento de la aplicación desde la perspectiva del usuario final.

9.1. Página de inicio

La página de inicio es la que recibirá a los usuarios, donde encontrarán una explicación de cómo funciona la aplicación (figura 25).

9.2. Registro

Los usuarios pueden registrarse con su correo electrónico a partir de la página de registro de la figura 26.

9.3. Inicio de sesión

Los usuarios ya registrados pueden iniciar sesión con su correo electrónico y contraseña en la página de inicio de sesión de la figura 27

9.4. Prepara tu viaje

La página *Prepara tu viaje* permite al usuario construir un itinerario personalizado de forma manual, seleccionando puntos de interés (POIs) y organizándolos por días. Dispone de múltiples opciones de uso, adaptadas a distintos niveles de personalización.

Al acceder a la vista inicial, la lista de POIs se encuentra vacía hasta que el usuario seleccione un destino o utilice su ubicación actual, como se muestra en la figura 1.

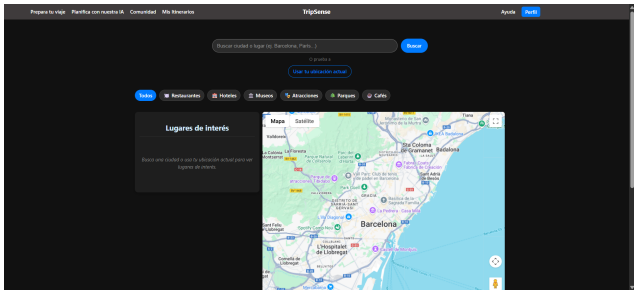

Fig. 1: Página Prepara tu viaje

En cuanto un usuario seleccione un destino, el mapa mostrará una lista con los puntos de interés cercanos (figura 2).

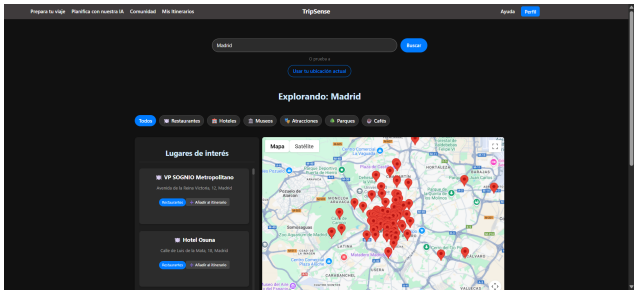

Fig. 2: Destino seleccionado

Si el usuario opta por utilizar su ubicación actual, se solicitará primero el permiso correspondiente (figura 3). Tras ser concedido, se mostrará la posición actual en el mapa (figura 4).

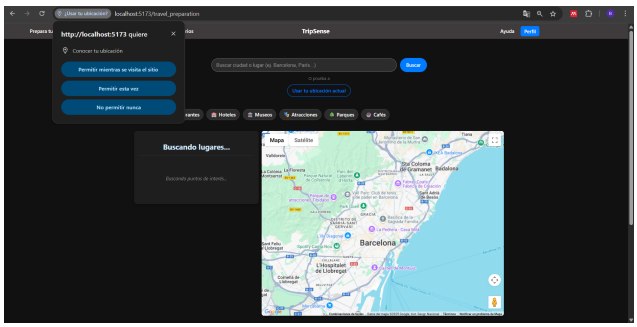

Fig. 3: Permiso para ubicación

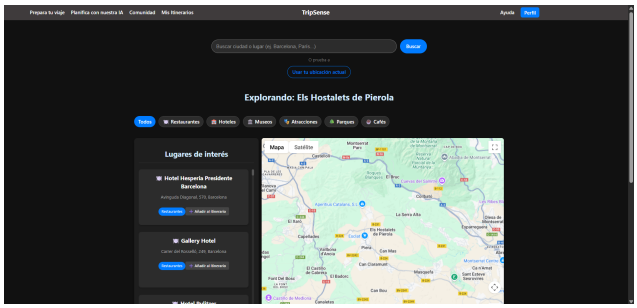

Fig. 4: Ubicación actual

La lista de POIs puede filtrarse por categorías (restaurantes, monumentos, museos, etc.), lo que permite al usuario acotar la búsqueda según sus intereses.

Una vez seleccionado el destino, el sistema habilita la opción de especificar las fechas del viaje. En función del rango de fechas introducido, se generan automáticamente pestañas correspondientes a cada día, facilitando la organización cronológica de las actividades (figura 5).

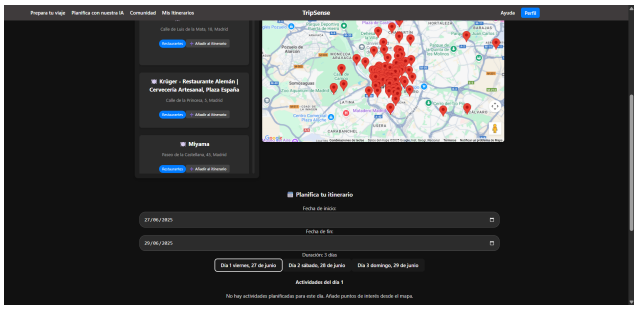

Fig. 5: Elección de duración del viaje

Cuando se añadan al menos un punto de interés en un día, se activará el botón de guardar itinerario. También el usuario puede especificar las horas de inicio y final para cada actividad y los medios de transporte que tendrá disponibles para poder calcular la ruta entre actividades (figura 6).

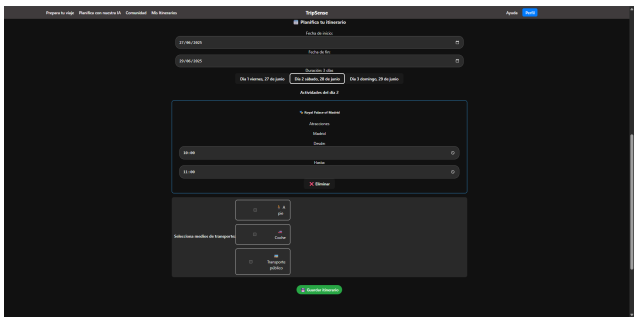

Fig. 6: Actividades divididas por días

9.5. Mis itinerarios

La pantalla *Mis itinerarios* permite al usuario consultar todos los itinerarios que ha guardado previamente. Cada entrada del listado muestra información esencial como el nombre del itinerario, el destino y las fechas del viaje. Desde esta vista, el usuario puede acceder a tres acciones principales: visualizar el itinerario en detalle, editarlo o eliminarlo (figura 28).

9.6. Detalle de itinerario

Al acceder a la vista detallada de un itinerario, se presenta toda la información estructurada del mismo: nombre, ciudad de destino, fechas del viaje, distancia total estimada, duración agregada, y medios de transporte seleccionados. Esta vista también ofrece botones para editar, eliminar o publicar el itinerario en la comunidad, así como una navegación por pestañas para explorar las actividades planificadas por día (figura 7).

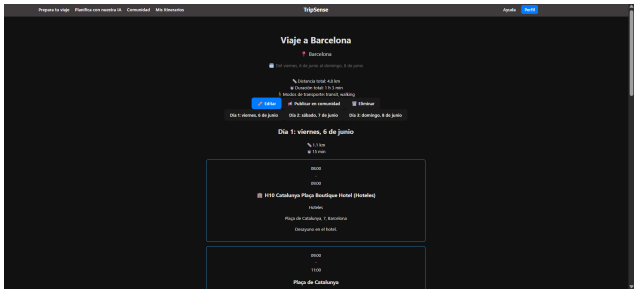

Fig. 7: Detalle de itinerario

Si el usuario decide compartir el itinerario con toda la comunidad, puede hacerlo presionando el botón de *Publicar en comunidad*. Esta acción abre un modal en la misma página, donde se muestra un mensaje informativo sobre el carácter público de la publicación, los datos que serán visibles para otros usuarios y un campo opcional para añadir una descripción (figura 8).

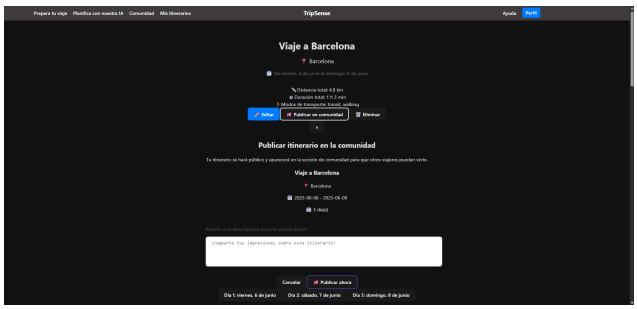

Fig. 8: Crear publicación de un itinerario

Al final de la lista de actividades se muestra un mapa con la ruta óptima calculada entre cada punto de interés. Este mapa muestra las rutas entre las actividades según el día que esté seleccionado (figura 9).

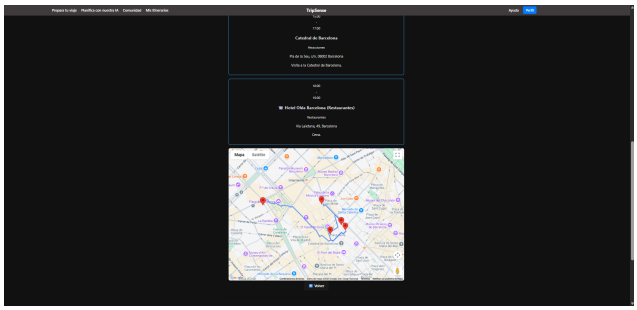

Fig. 9: Mapa con las rutas por días

9.7. Edición de itinerario

La pantalla de edición de itinerario permite al usuario modificar cualquier aspecto del itinerario previamente creado o generado. La interfaz muestra la lista de actividades organizadas por días, similar a la vista de detalle, pero con funcionalidades adicionales de edición.

Cada actividad incluye un botón específico para eliminarla del itinerario (figura 10). Por encima de la lista diaria, se encuentra el botón *Añadir actividad*, que al ser pulsado abre un modal con un catálogo de puntos de interés obtenidos dinámicamente para la ciudad asociada al itinerario en edición. Desde este catálogo, el usuario puede seleccionar nuevas actividades para incorporarlas al plan de viaje.

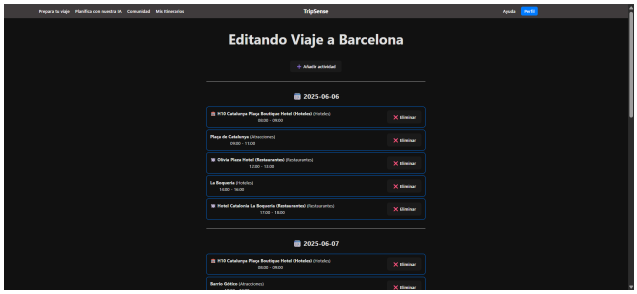

Fig. 10: Edición de itinerario

Seguidamente de la lista de días, aparece la opción de editar los modos de transporte disponibles. Cuando se modifican estas propiedades, se vuelven a calcular automáticamente las rutas óptimas por día entre los puntos de interés y, finalmente, el botón de guardar (figura 11).

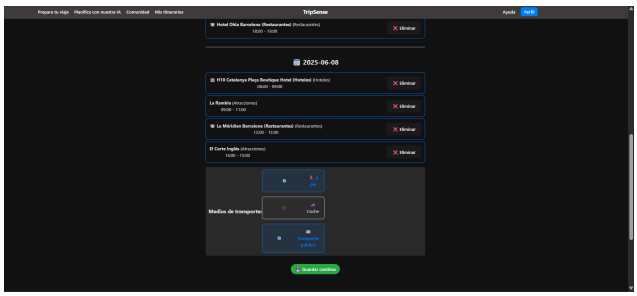

Fig. 11: Edición de itinerario

9.8. Generador de itinerarios con IA

Cuando el usuario entra en la página de *Generador de itinerarios con IA* se encontrará con un formulario a rellenar. En este formulario, el usuario debe indicar el destino, el cual se basará en el itinerario a generar por el modelo de lenguaje. La siguiente información a facilitar son las fechas de inicio y final del viaje; en estos campos se abrirá un calendario para ayudar al usuario visualmente.

A continuación, se encuentran unos botones que permiten seleccionar varias opciones, representando los medios de transporte. El usuario deberá elegir como mínimo uno para poder realizar el cálculo de rutas.

Por último, el usuario, opcionalmente, puede añadir alguna especificación adicional para que la inteligencia artificial lo tenga en cuenta. Como, por ejemplo, pedir que añada algún punto de interés en concreto. El estado inicial de la pantalla se encuentra en la figura 12.

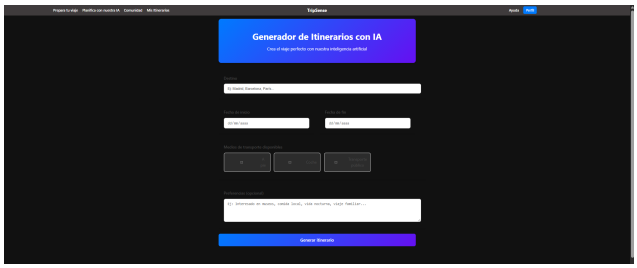

Fig. 12: Página de generador de itinerarios con IA

En la figura 13 se muestra el formulario completamente rellenado, listo para su envío al modelo de inteligencia artificial.

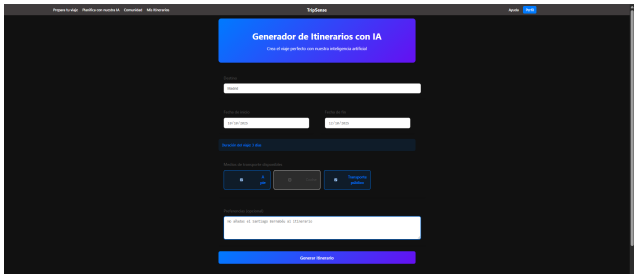

Fig. 13: Formulario rellenado

A continuación se muestra en la figura 14 el resultado obtenido al presionar el botón de *Generar itinerario*. Se observa una lista dividida por días compuesta de puntos de interés que muestran su nombre, franja horaria para visitar y dirección de cada uno.

Una vez generado, el usuario puede decidir entre guardar el itinerario para incorporarlo a su lista personal o volver a generar otro, descartando la propuesta actual.

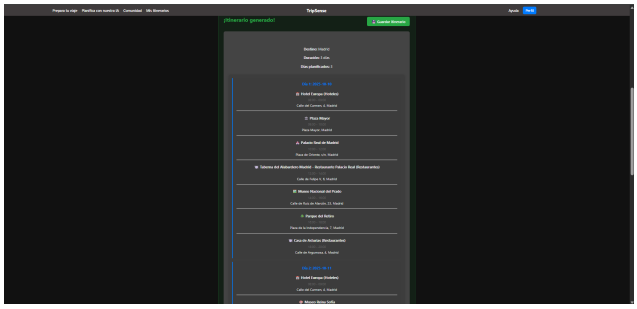

Fig. 14: Itinerario generado por inteligencia artificial

9.9. Perfil de usuario

En la figura 15 se muestra cómo un usuario vería su perfil propio después de iniciar sesión y pulsar el botón *Perfil*.

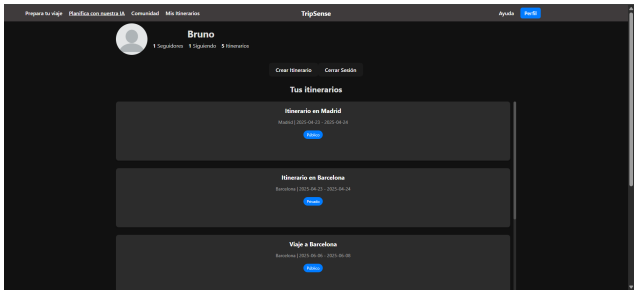

Fig. 15: Perfil de usuario propio

En la cabecera de la página se muestran datos relevantes como su nombre de usuario, número de seguidores, número de perfiles seguidos y número de itinerarios. Si el perfil mostrado es el propio, el número de itinerarios corresponde al total de itinerarios guardados, públicos o privados. En cambio, si se trata del perfil de otro usuario, únicamente se muestran los itinerarios publicados públicamente.

Debajo de los datos del usuario hay la opción de crear un itinerario, el cual redirige a la pantalla de *Preparación de viaje* (9.4), y la opción de cerrar sesión.

Seguidamente, se expone una lista con los itinerarios, que contiene los guardados por el usuario o, si visita otro perfil, los itinerarios publicados.

La figura 16 muestra un ejemplo de visualización de un perfil ajeno. En este caso, se omiten las acciones de creación de itinerarios y cierre de sesión, y se presenta un botón de *Seguir / Dejar de seguir* que permite establecer o eliminar una relación social entre usuarios.

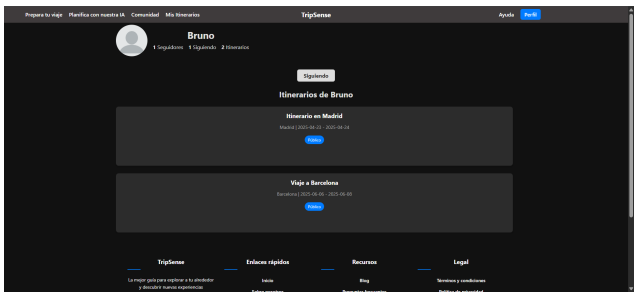

Fig. 16: Perfil de otro usuario

9.10. Feed de comunidad

.

En la figura 17 se muestra la interfaz del feed, que presenta una lista de publicaciones ordenadas cronológicamente, mostrando quién es el autor, la fecha de publicación, una descripción creada por el usuario y los datos generales del itinerario, que son título, destino y fechas que lo componen.

Si se presiona el botón de *Ver más* se amplía la información mostrando la duración total en días del itinerario y la distancia acumulada entre las actividades del viaje.

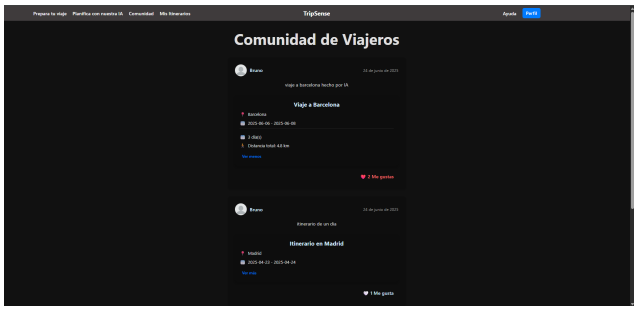

Fig. 17: Página de comunidad

10 Conclusiones

El desarrollo de este trabajo final de grado ha permitido materializar una aplicación web funcional que combina recomendaciones geolocalizadas con funcionalidades sociales e inteligencia artificial, orientada a mejorar la planificación personalizada de viajes. Este proyecto ha perseguido desde el inicio unos objetivos claros: diseñar una plataforma accesible, útil y flexible que permita a los usuarios crear itinerarios adaptados a sus preferencias y necesidades, explorando puntos de interés cercanos, compartiendo experiencias con la comunidad y obteniendo asistencia automatizada mediante IA.

Durante el transcurso del trabajo se han abordado con éxito los objetivos propuestos, superando retos tanto técnicos como de diseño. Entre los principales logros destacan: la integración de la API de Google Maps para geocodificación, cálculo de rutas y obtención de puntos de interés; el desarrollo de una infraestructura backend robusta en Node.js con TypeScript; la implementación de funcionalidades sociales; y la incorporación de un generador de itinerarios basado en modelos de lenguaje de última generación disponibles con la API de *Google AI for developers*.

Sin embargo, cabe destacar que, como en todo trabajo técnico, no todos los objetivos se han alcanzado. Inicialmente se planteó como objetivo el desarrollo de un sistema

de recomendaciones personalizadas de actividades para cada usuario, basadas en su historial y preferencias. Esta funcionalidad no ha sido implementada y queda identificada como una línea de trabajo futuro. Asimismo, otras mejoras a realizar serían el soporte multilingüe y el desarrollo específico para dispositivos móviles.

REFERENCIAS

[1] "10 apps que te harán la vida más fácil cuando estés de viaje - bbc news mundo," fecha de acceso: 15/06/2025. [Online]. Available: https://www.bbc.com/mundo/noticias-37546893

[2] "Komoot — descubre, planifica y comparte tus aventuras con komoot," fecha de acceso: 15/06/2025. [Online]. Available: https://www.komoot.com/es-es

[3] "Strava — quiénes somos," fecha de acceso: 15/06/2025. [Online]. Available: https://www.strava.com/about

[4] J. Martins, "¿qué es la metodología kanban y cómo funciona? [2025] • asana," 1 2025, fecha de acceso: 24/02/2025. [Online]. Available: https://asana.com/es/resources/what-is-kanban

[5] J. G. Maheshwari, V. S. Gogi, I. N. Aradhya, and Shankaramma, "Digitalization of lean tools and techniques: Enhancing efficiency and quality in modern industries," *8th IEEE International Conference on Computational System and Information Technology for Sustainable Solutions, CSITSS 2024*, 2024.

[6] "Jira — software de seguimiento de proyectos e incidencias — atlassian," fecha de acceso: 24/02/2025. [Online]. Available: https://www.atlassian.com/es/software/jira

[7] "Github," fecha de acceso: 24/02/2025. [Online]. Available: https://github.com/

[8] "React," fecha de acceso: 24/02/2025. [Online]. Available: https://react.dev/

[9] "Typescript: Javascript with syntax for types." fecha de acceso: 14/06/2025. [Online]. Available: https://www.typescriptlang.org/

[10] "Css — mdn," fecha de acceso: 14/06/2025. [Online]. Available: https://developer.mozilla.org/es/docs/Web/CSS

[11] "React router home — react router," fecha de acceso: 14/06/2025. [Online]. Available: https://reactrouter.com/home

[12] "Axios," fecha de acceso: 09/04/2025. [Online]. Available: https://axios-http.com/es/

[13] "Getting started — vite," fecha de acceso: 14/06/2025. [Online]. Available: https://vite.dev/guide/

[14] "Node.js — ejecuta javascript en cualquier parte," fecha de acceso: 24/02/2025. [Online]. Available: https://nodejs.org/es

[15] "Express - node.js web application framework," fecha de acceso: 14/06/2025. [Online]. Available: https://expressjs.com/

[16] W. Bastidas, "Estructura de una api rest con nodejs, express y mongodb," 2 2018. [Online]. Available: https://medium.com/williambastidasblog/estructura-de-una-api-rest-con-nodejs-express-y-mongodb-cdd97637b18b

[17] S. Khedkar and S. Thube, "Real time databases for applications," *International Research Journal of Engineering and Technology*, 2017. [Online]. Available: www.irjet.net

[18] "Mongoose v8.15.2: Schemas," fecha de acceso: 15/06/2025. [Online]. Available: https://mongoosejs.com/docs/guide.html

[19] "Descripción general — maps javascript api — google for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://developers.google.com/maps/documentation/javascript/overv 419

[20] "Documentación de google maps platform — geocoding api — google for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://developers.google.com/maps/documentation/geocoding?hl=e 419

[21] "Documentación de google maps platform — places api — google for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://developers.google.com/maps/documentation/places/web-service?hl=es-419

[22] "Cómo obtener instrucciones sobre cómo llegar con la api de directions — directions api — google for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://developers.google.com/maps/documentation/directions/get-directions?hl=es-419

[23] "Modelos de gemini — gemini api — google ai for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/models?hl=es-419authuser=1gemini-2.0-flash

[24] A. Fenollosa, "Algoritmo de mezcla fisher-yates — programador web valencia," 4 2024, fecha de acceso: 23/05/2025. [Online]. Available: https://programadorwebvalencia.com/algoritmo-de-mezcla-fisher-yates/

[25] "Resultados estructurados — gemini api — google ai for developers," fecha de acceso: 23/05/2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/structured-output?hl=es-419authuser=1

APÉNDICE

A.1. Prototipo figma

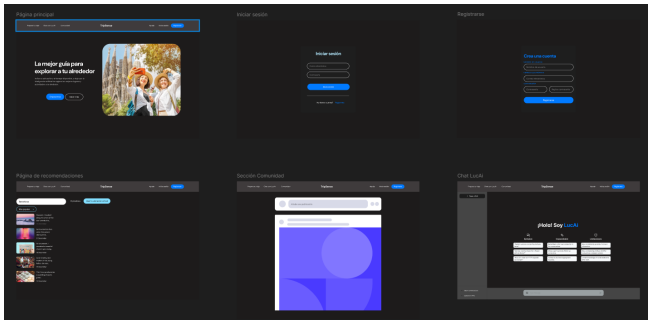

Fig. 18: Diseño del prototipo en Figma

A.2. Diagrama de casos de uso

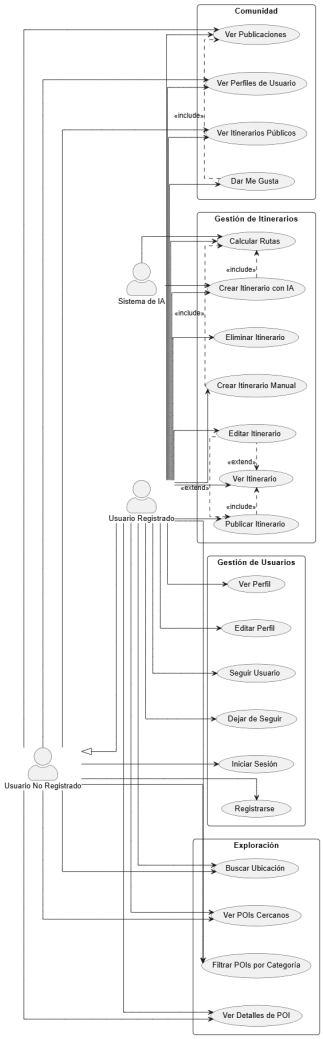

Fig. 19: Diagrama de casos de uso

A.3. Diagrama de base de datos

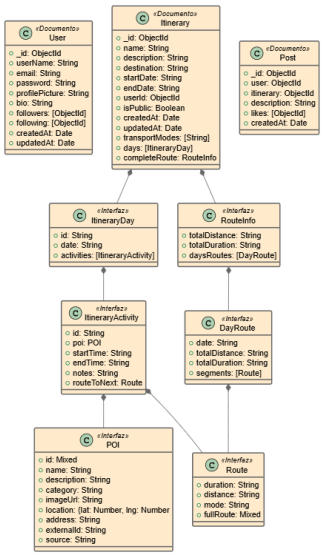

Fig. 20: Diagrama base de datos

A.4. Esquema JSON

```
// Configuración del cliente de Gemini
const ai = new GoogleGenAI({ apiKey: process.env.GOOGLE_GENAI_API_KEY });

// Esquema para la respuesta de Gemini
const itineraryschema = {
    type: Type.OBJECT,
    properties: {
        days: {
            type: Type.ARRAY,
            items: {
                type: Type.OBJECT,
                properties: {
                    id: {
                        type: Type.STRING,
                        description: "ID único del día"
                    },
                    date: {
                        type: Type.STRING,
                        description: "Fecha en formato YYYY-MM-DD"
                    },
                    activities: {
                        type: Type.ARRAY,
                        items: {
                            type: Type.OBJECT,
                            properties: {
                                id: {
                                    type: Type.STRING,
                                    description: "ID único de la actividad"
                                },
                                poi: {
                                    type: Type.OBJECT,
                                    properties: {
                                        id: {
                                            type: Type.NUMBER,
                                            description: "ID numérico del punto de interés"
                                        },
                                        name: {
                                            type: Type.STRING,
                                            description: "Nombre del lugar con emoji apropiado"
                                        },
                                        description: {
                                            type: Type.STRING,
                                            description: "Dirección o descripción del lugar"
                                        },
```

Fig. 21: Esquema JSON de respuesta

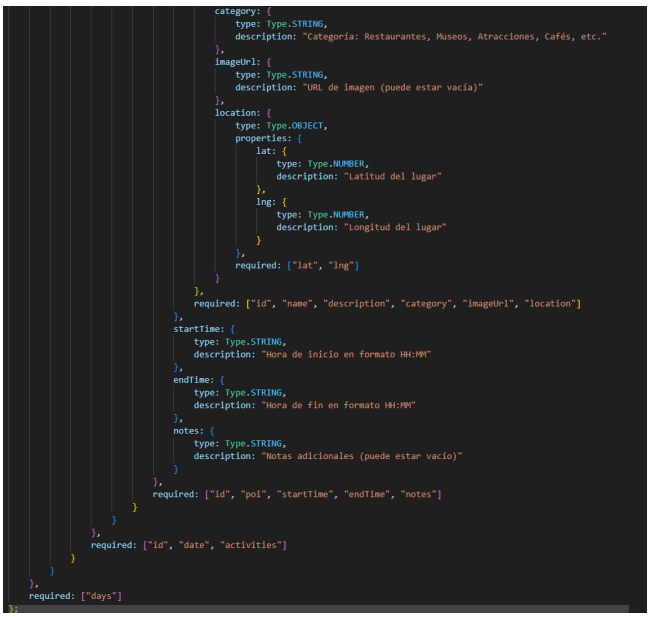

Fig. 22: Esquema JSON de respuesta

A.5. Definición de interfaz POI

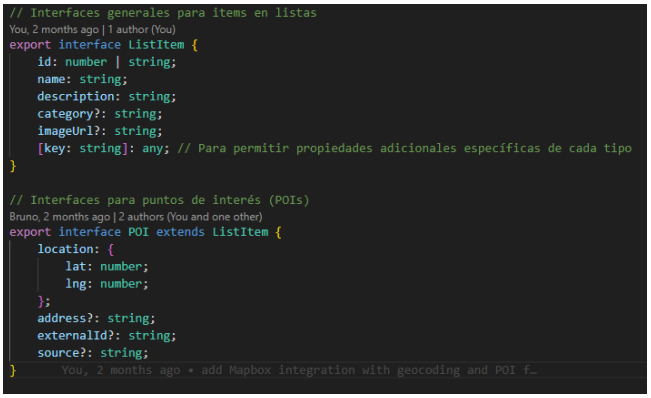

Fig. 23: Definición Point Of Interest

A.6. Prompt para el modelo

Fig. 24: Prompt dinámico para el modelo de lenguaje

A.7. Resultados generales

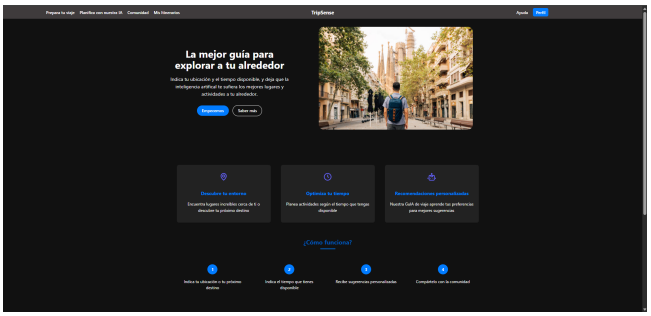

Fig. 25: Página de inicio

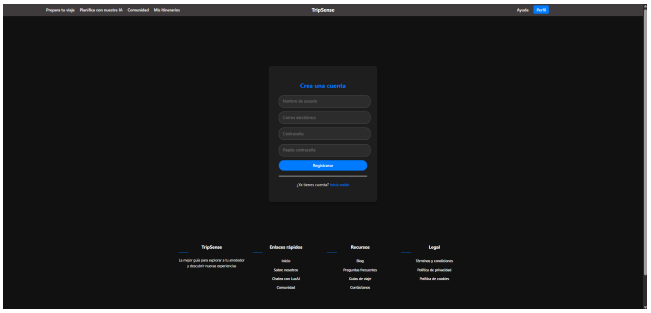

Fig. 26: Registro

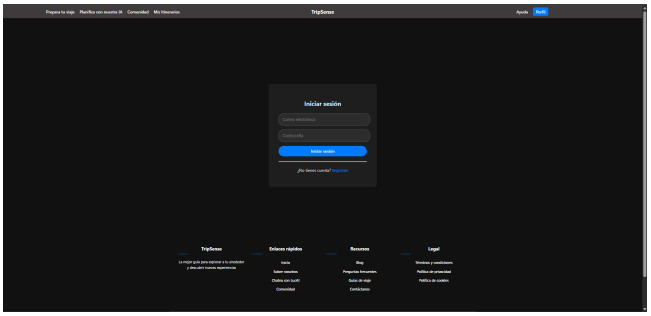

Fig. 27: Inicio de sesión

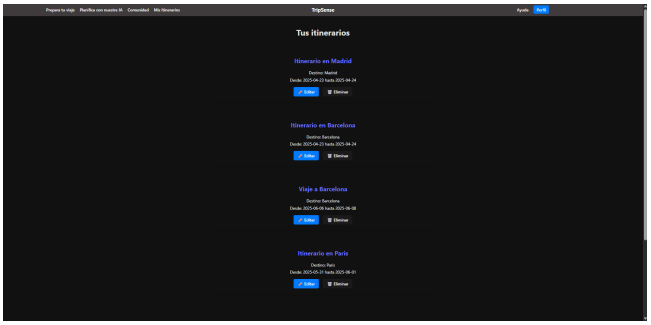

Fig. 28: Página Mis itinerarios