UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación móvil para la recomendación de sitios cercanos basándose en gustos del usuario

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Benelkadi, Mehdi

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2023/2024

Resum

L'objectiu d'aquest treball és el desenvolupament d'una aplicació la funció de la qual és la recomanació de llocs propers basats en els gustos de l'usuari, la seua posició i el seu temps disponible. Per a això, l'aplicació permet la creació d'un compte d'usuari perquè l'usuari puga registrar els seus gustos personals i les seues aficions. La cerca de llocs es farà mitjançant l'API de Google Maps. L'usuari podrà interactuar amb els llocs recomanats per a deixar una nota, un comentari i se li donarà l'opció de guardar els seus llocs favorits.

Paraules clau: Frontend, Backend, API, React Native, Node.js, Postman, HTTP, Servidor, Google Maps, Aficions, Etiquetes, Usuari, Aplicació, Web, Compte, Contrasenya, Llocs, Pantalla, Component, Petició, Base de Dades, SQL, MySQL, Interfície, Perfil, Validació

Resumen

El objetivo de este trabajo es el desarrollo de una aplicación cuya función la recomendación de sitios cercanos basados en los gustos del usuario, su posición y su tiempo disponible. Para ello la aplicación permite la creación de una cuenta de usuario para que el usuario pueda registrar sus gustos personales y sus aficiones. La búsqueda de sitios se hará mediante la API de Google Maps. El usuario podrá interactuar con los sitios recomendados para dejar una nota, un comentario y se le dará la posibles de guardar sus sitios favoritos.

Palabras clave: Frontend, Backend, API, React Native, Node.js, Postman, HTTP, Servidor, Google Maps, Aficiones, Tags, usuario, aplicación, Web, Cuenta, Contraseña, Sitios, Pantalla, Componente, petición, Base de Datos, SQL, MySQL, interfaz, perfil, validación

Abstract

The goal of this project is the development of an application whose function is to recommend nearby locations based on the user's preferences, their position, and available time. To achieve this, the application allows the creation of a user account so the user can register their personal preferences and hobbies. The search for locations will be carried out using the Google Maps API. The user will be able to interact with the recommended locations to leave a rating, a comment, and will be given the option to save their favorite locations.

Key words: Frontend, Backend, API, React Native, Node.js, Postman, HTTP, Server,Google Maps, Hobbies, Tags, User, Application, Web, Account, Password, Locations, Screen, Component, Request, Database, SQL, MySQL, Interface, Profile, Validation

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Nuestras ciudades están llenas de sitios interesantes que solo esperan a ser descubiertos. Si a veces cuesta encontrar sitios interesantes en su propia ciudad, se vuelve aún más difícil cuando se habla de viajes a otra ciudad, o incluso a otro país. Para intentar aportar una solución a este problema, nació esta idea de aplicación que se basa en un principio muy sencillo. Un usuario que se registra, al crear su perfil, también está invitado a registrar sus aficiones y los sitios que le interesaría descubrir. Luego, al abrir la aplicación, el usuario introduce el tiempo que está dispuesto a gastar. De esta manera, en situaciones en las que una persona está esperando una cita, un tren, o cualquier otra cosa, y esté en un sitio que no conoce, puede aprovechar para descubrir el barrio o hasta una nueva ciudad. Basándose en el tiempo disponible y en los gustos del usuario, la aplicación devolverá una lista de sitios cercanos que pueda interesar al usuario. Utilizando tecnologías modernas como React Native para ofrecer una interfaz intuitiva y sencilla de usar, y la API de Google Maps para poder recuperar informaciones de distintos sitios, esta aplicación permitirá tanto poner en luz sitios escondidos e interesantes como descubrir su propio barrio o hacer turismo de una manera diferente. En este contexto, se ha desarrollado una solución sencilla de usar y eficiente, con tecnologías modernas como viene a ser React Native, framework líder en el desarrollo de aplicaciones híbridas.

1.1 Motivación

La idea detrás de esta aplicación viene de un día en el que me retrasaron una cita médica de una hora a último momento, ya estando en la sala de espera. Al estar presente, no me quedaba otra opción que esperar mi turno, dando una vuelta por el barrio. Ahí es cuando se me ocurrió la idea de esta aplicación. Con ella podría haberme enterado de sitios cercanos que me podrían haber interesado para pasar el tiempo, y que no conocía, y aprovechar el tiempo que tenía para descubrir nuevos sitios en mi ciudad. Naturalmente, la plataforma más adaptada a este tipo de aplicaciones es la plataforma móvil. Y para una mayor flexibilidad, se usarán herramientas de desarrollo híbrido para soportar un máximo número de plataformas y de usuarios. De esta forma un mayor número de usuarios podrán disfrutar de las funcionalidades de la app, mientras cuentan con una cobertura 4G.

1.2 Objectivos

El objetivo general del proyecto consiste en el desarrollo de una aplicación móvil que cuenta con funcionalidades de localización para poder recuperar las posiciones de

los distintos sitios de interés y del usuario y guiarlo a través de la ciudad hacía dichos sitios. Al ser una aplicación dirigida al público, también se tiene que tener en cuenta el desarrollo de una interfaz sencilla de entender y usar para poder permitir la mayor accesibilidad a todos los usuarios. Para conseguir este objetivo se plantean los siguientes objetivos específicos :

- La aplicación devuelve un listado de sitios basados en los gustos del usuario y en su tiempo disponible.
- El usuario puede crearse un perfil.
- En su perfil, el usuario puede registrar sus gustos, aficiones e intereses.
- El usuario podrá cualificar los sitios recomendados y dejar comentarios. Los comentarios serán visibles para todos los usuarios. El usuario debe introducir el tiempo que tiene disponible.

1.3 Estructura de la memoria

1. Introducción : Parte actual en la que se ha introducido el trabajo, su motivación, sus objetivos y la estructura de la memoria.
2. Estado del arte : Presentación de otras aplicaciones existentes con un objetivo o un funcionamiento similar.
3. Metodología : Metodología incremental para el diseño y desarrollo de la aplicación.
4. Análisis de requisitos : Identificación de los requisitos que debe soportar la aplicación mediante cuestionarios, casos de uso y escenarios.
5. Análisis conceptual y diseño : Diseño de la aplicación mediante bocetos de las interfaces y diseño del modelo de base de datos.
6. Desarrollo de la solución : Presentación de las herramientas y frameworks que se van a usar y ejemplos de código.
7. Presentación del producto final.

CAPÍTULO 2
Estado del arte

Antes de empezar con el desarrolle la solución, es interesante ver otras aplicación cuyo objetivo es similar al de este proyecto o que embarcan funciones parecidas a lo que se quiere desarrollar. De esta manera se podrá ver ejemplos de arquitecturas funcionales, de errores a evitar o simplemente definir el valor añadido de la aplicación frente a las disponibles en el mercado.

2.1 Google Maps

Lógicamente, si la aplicación a desarrollar utilizará la API de Google Maps para poder funcionar, significar que Google Maps en si tiene funcionalidades interesantes que permiten llegar al mismo resultado de otra manera. Google Maps es famosa para ser una aplicación de mapa/gps que permite llegar guiarse en el día día. Esto es cierto, pero sería dejar una gran parte de todo lo que puede ofrecer.Además de las funciones tradicionales de navegación, Google Maps permite a los usuarios buscar lugares según sus intereses específicos, como restaurantes, parques, eventos o tiendas, y personalizar sus rutas o recomendaciones según sus gustos. Con herramientas como la vista de calle, o Street View, alertas de tráfico en tiempo real, y la capacidad de guardar ubicaciones favoritas, Google Maps no solo sirve para llegar de un lugar a otro, sino también para descubrir nuevas experiencias y personalizar las interacciones con el entorno según las necesidades y aficiones del usuario. Asimismo, Google Maps permite a los usuarios interactuar de forma activa con su entorno: pueden dejar reseñas, calificar lugares, añadir fotos y sugerir modificaciones, lo que contribuye a una experiencia más rica y personalizada. Estas características hacen de Google Maps no solo una herramienta de navegación, sino una potente plataforma de descubrimiento y exploración. Dentro de todas estas funcionalidades, las más interesantes en el contexto de este trabajo son la posibilidad de buscar lugares según los intereses del usuario, guardar las ubicaciones de interés y la posibilidad de dejar reseñas.

Lo que le diferencia del proyecto que se ha de desarrollar es la necesidad muchas interacciones del usuario para poder encontrar un sitio. Aquí para encontrar un sitio de algún interés, cuando no sé sabe exactamente lo que se busca, hay primero que describir algún tipo de sitio, como 'restaurante', 'librería' o 'tienda de discos' para luego ver todos los sitios devueltos por la aplicación. El objetivo del proyecto es partir del mismo principio, para devolver directamente los sitios basándose en con un numero mínimo de interacción por le usuario.

2.2 Aiplan.es

Aiplan es una web que permite buscar planes, solo o entre amigos, según aglunos criterios decididos por el usuario. Los planes se pueden buscar por categorías como juegos, deporte, cine, escape room, o al aire libre. Dispone de una pagina de 'trending' para ver los mejores planes disponibles. Las actividades son muy detalladas y permiten elegir los sitos por zona, precios, numero de personas o tiempo de estancia. Es muy completa tiene varías opciones interesantes. Se presenta la pagina de incio en la figura 2.1.

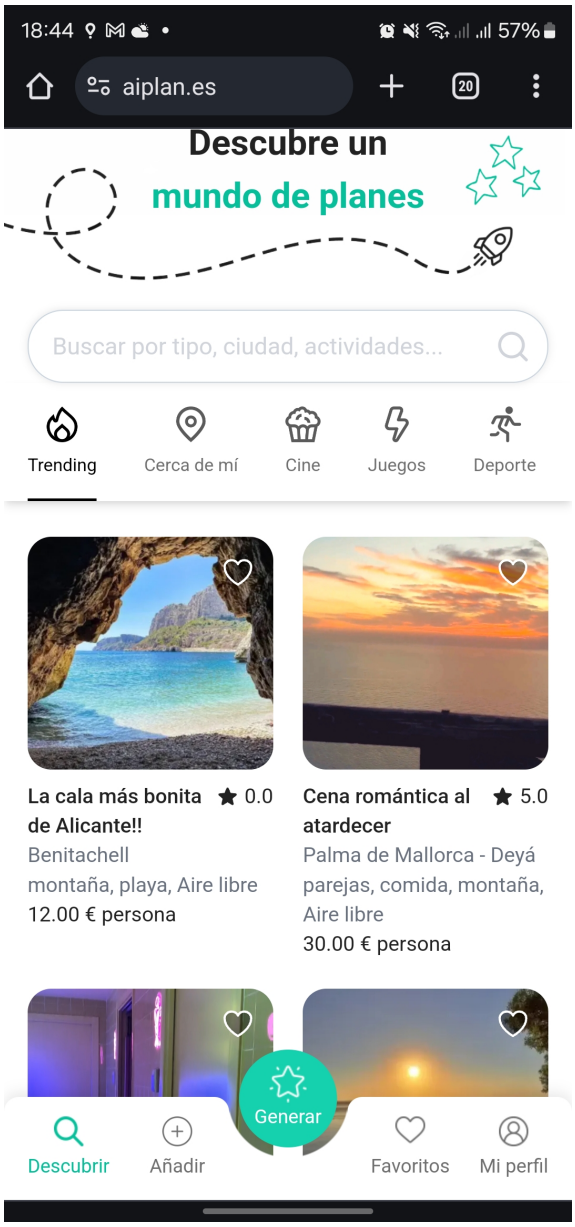

Figura 2.1: Aiplan.es - Inicio

Además de proponer planes muy variados, los usuarios tiene la opción de generar un plan. Tras contestar a un par de preguntas como se muestra un ejemplo en la figura 2.2, la web generará un plan acorde a las respuestas del usuario . El cuestionario es muy exhaustivo, y se pide informar sobre el numero de personas, la franja horaria, el modo de desplazamiento, el tiempo máximo para llegar etc... Además se permite dejar

comentarios y calificaciones a los sitios para poder ayudar a los demas usuarios en su elección.

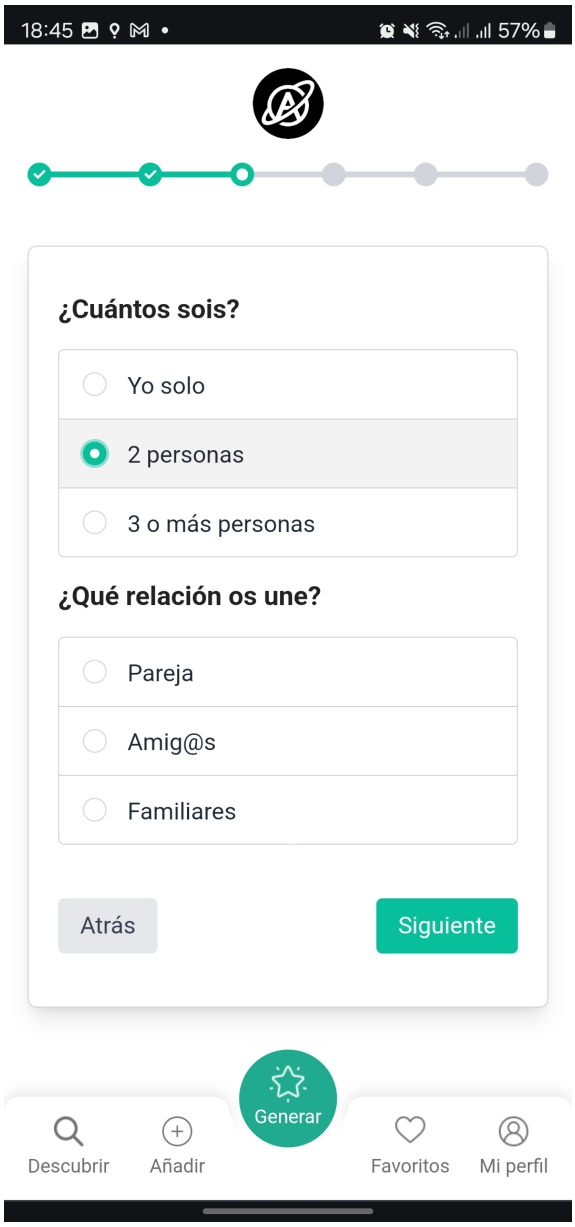

Figura 2.2: Aiplan.es - Generación de Planes

2.3 Visit A City

Visit a City es una herramienta diseñada para ayudar a los turistas planificar sus visitas y descubrir diferentes ciudades a través del mundo de manera eficiente y personalizada. Ofrece varios itinerarios, basados en el tiempo de estancia en la ciudad y el tipo de actividad. La aplicación tiene varios tipo de sugerencias de atracciones turísticas, museos, restaurantes y actividades populares en la ciudad. Además de eso, permite personalizar las diferentes rutas para ajustarse aún más a los gustos del usuario. En caso de no disponer de conexión, ofrece también un mapa disponible en modo offline. Al entrar en la aplicación se pide la ciudad en la que se va a viajar en los próximos días. Una vez la ciudad seleccionada Se dispone de varias opciones.

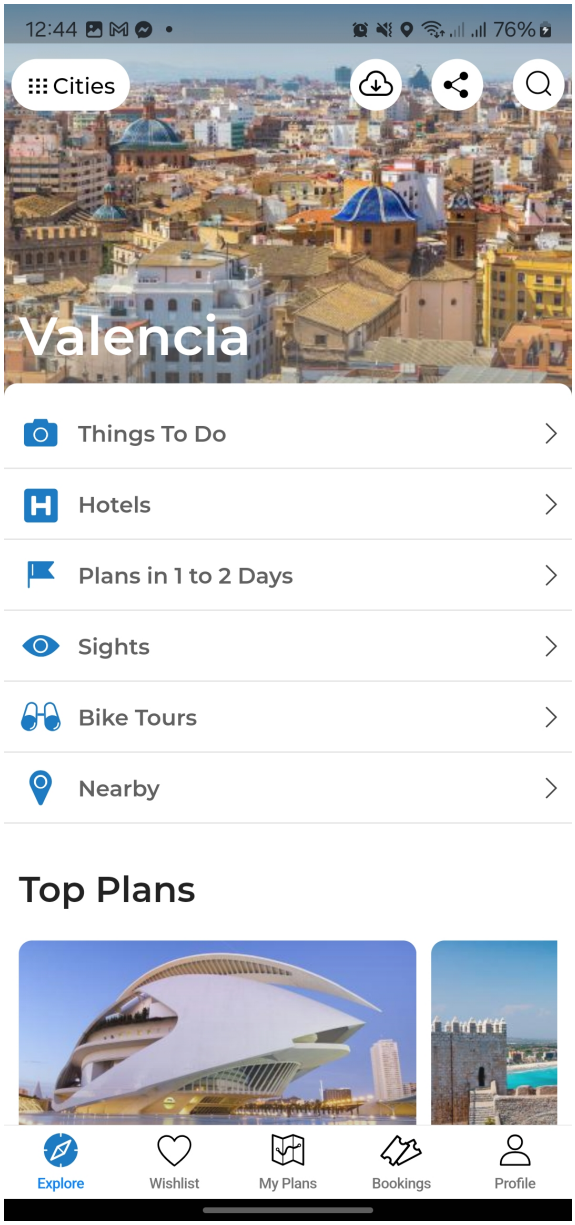

Figura 2.3: Visit a City - Inicio

En la figura 2.3 se puede ver las distintas opciones de planificación y de navegación en la aplicación. Se dispone de varías opciones que van de la lista de sitios cercanos a reservas de hotel. También se ve un apartado 'Top Plans' que lista los planes que más interesantes y más completos en la ciudad. El apartado 'Things To Do', es lo que más se podría parecer a lo que se quiere desarrollar con el proyecto. Aquí se puede elegir actividades por tipo, fecha, precio o duración, como lo muestra la figura 2.4.

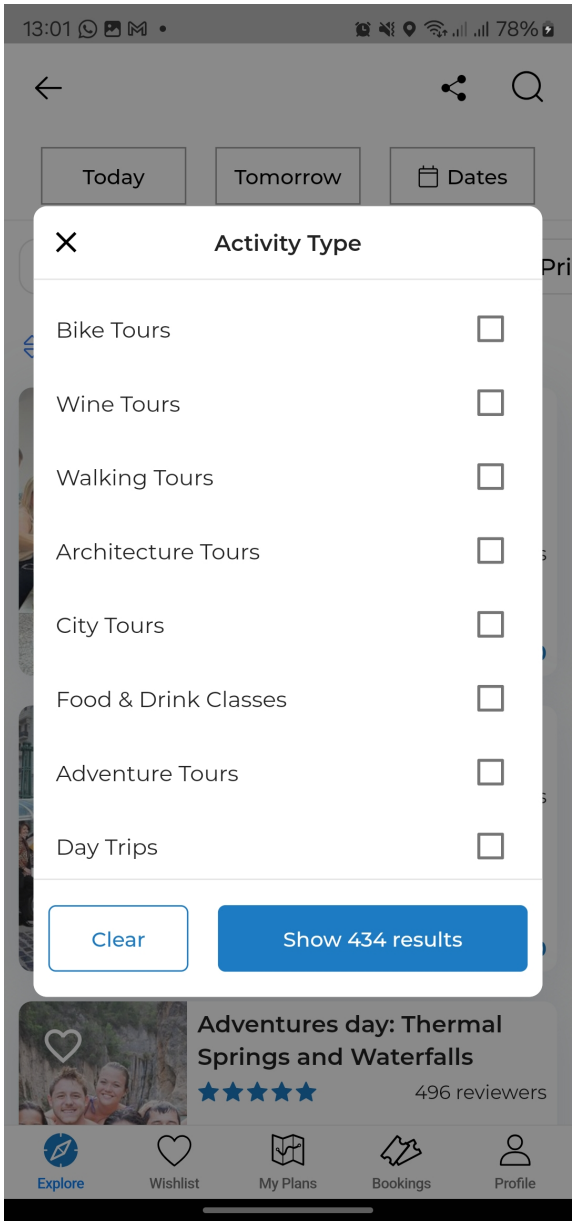

Figura 2.4: Visit a City - Things to do

Esta aplicación facilita la planificar de viaje y permite hacer turismo pero no está adaptada a un uso cotidiano. Además, las actividades propuestas son actividades largas que ocupan una tarde entera, sino mas. Si en su forma se puede parecer al proyecto a desarrollar, los objetivos finales son muy distintos. El proyecto está pensado para un uso más simple, y más tierno. No se busca hacer turismo, simplemente descubrir sitios de interés en un hueco de unos minutos a una hora en un día, lo cual es bastante diferente. Por otro lado, las actividades propuestas por 'Visit a City' son actividades exclusivamente turísticas. No sé puede simplemente buscar un sitio en concreto, lo que se puede mirar en la mayoría de veces son actividades.Los usuarios finales de ambas aplicaciones son muy distintos.

2.4 Identify

Identify es una aplicación cuyo funcionamiento es muy sencillo. Tiene la misma interfaz que Google Maps y permite encontrar sitios culturales en una ciudad. Al crear un perfil, se le pide al usuario elegir los que le interesa ver (Figura 2.5). Se permite entre elegir entre varios sitios culturales. Según lo que se ha elegido, se mostrará en el mapa sitios que correspondientes.

Figura 2.5: Identify - Creación del Perfil

Además de esto, Identify tiene una funcionalidad muy interesante que permite crear rutas. Cada ruta tiene un nombre y una descripción, para informar del contenido de la ruta. Las rutas pasan por varios sitios de interés y permite tener un especie de vista guiada sin la necesidad de tener a un experto al lado.

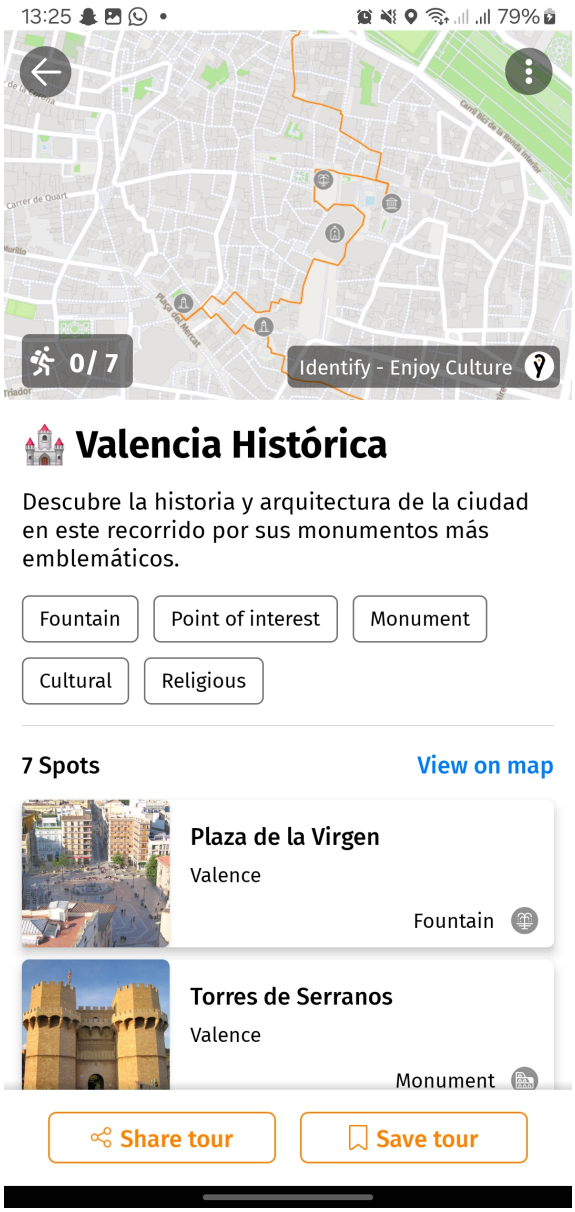

Figura 2.6: Identify - Ejemplo de ruta

Por ejemplo, en la figura 2.6 se muestra una ruta, llamada 'Valencia Histórica', que recorre todos los sitios históricos del centro de la ciudad. Por otra parte, en la aplicación se ha hecho un esfuerzo para poner en el centro las interacciones entre usuarios. Las rutas no fueron generadas por la aplicación, sino por los usuarios, además de poder registrar sitios de interes.. En el perfil, se da la opción de crear una ruta, poner una descripción y guardar su recorrido. Estas rutas son publicas, y para animar a los usuarios a crear rutas, existe un sistema de puntos y de seguidores para los perfiles aprovechando al máximo esta funcionalidad. Cuanto más usuarios pasaron por una ruta, más puntos gana su creador. Los perfiles con más contribución a la aplicación aparecen en un ranking general de la región, como lo muestra la figura 2.7. Esto permite motivar a los usuarios a interactuar con la aplicación y registrar nuevos sitios y nuevas.

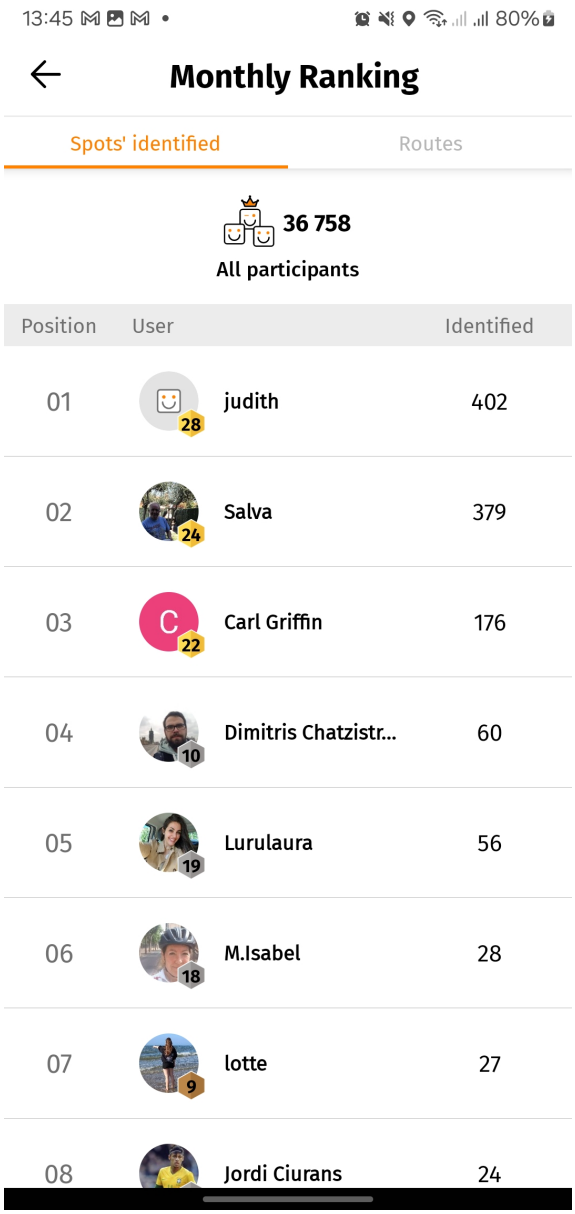

Figura 2.7: Identify - Ranking de usuarios con más contribución

La manera de conectar los usuarios entre ellos es una muy buena manera de crear interacciones y de hacer que la aplicación sea cada vez más completa. Está aplicación tiene como ventaja permitir a los usuario descubrir sitios culturales alrededor y hacer turismo de bario. No es necesario recuperar datos de ningún pero si que hace falta una base inicial de usuarios importante para que todos puedan disfrutar plenamente de las funcionalidades que ofrece la aplicación.

2.5 Solución Propuesta

Todas las aplicaciones vistas tienen ideas interesantes pero ninguna se adapta realmente al problema plantada al principio de este trabajo. Se busca más un pasatiempo de unas decenas de minutos que realmente crear planes de una tarde entera o varios días como lo proponen algunas aplicaciones. La idea es ofrecer una solución que permita a los usuarios descubrir actividades rápidas y espontáneas, algo que puedan disfrutar en

su tiempo libre sin la necesidad de una planificación anterior.El objetivo es desarrollar una herramienta que sugiera opciones inmediatas y accesibles según la ubicación actual y los intereses del usuario, algo más flexible y adaptable que los itinerarios completos o las actividades de larga duración que suelen ofrecer las aplicaciones tradicionales de planificación de viajes. La aplicación debe ser capaz de proporcionar recomendaciones inmediatas y adaptadas al contexto del usuario, como descubrir un café cercano, una galería que se pueda visitar en menos de una hora, o una tiendo de algo que sea de su interés. Este enfoque flexible podría permitir que el usuario disfrute de pequeños momentos de esparcimiento o descubrimiento sin la necesidad de una planificación.

CAPÍTULO 3
Metodología

Para el desarrollo de este proyecto, se ha usado una metodología llamada metodología incremental[3]. Consiste en dividir el desarrollo del proyecto en pequeños módulos que se prueban y entregan por separado de manera secuencial. Estos módulos se van integrando poco a poco para construir la aplicación hasta llegar a un producto entero. Es un método muy útil para el desarrollo de proyectos complejos, ya que nos aseguramos de que cada bloque individual funciona correctamente antes de integrarlo.

La metodología incremental tiene varías características y ventajas que vamos a detallar a continuación. El modelo incremental se basa en un desarrollo por etapas o bloques. Cada etapa se desarrolla por separado para asegurarse de que cada bloque añadido es funcional y no cause problemas de dependencias con el resto del proyecto. Este desarrollo por etapas permite tener un feedback constante con el cliente e involucrarlo en todo el proceso de desarrollo de la aplicación, no solo en momentos puntuales. De esta forma reducimos el número de alteraciones al proyecto por malentendidos con el cliente. Este tipo de desarrollo es mucho más flexible y adaptable. Facilita la incorporación de nuevas funcionalidades y cambios sin que sea necesario una reestructuración completa del proyecto. Una vez un bloque está acabado, probado, y validado, no debería de ser modificado salvo algunos casos específicos como pueden ser errores o fallos de compilación.

Para llegar a desarrollar un proyecto con el modelo incremental, hay un par de etapas previas al desarrollo de código que hay que cumplir para poder definir objetivos claros que aclaran el camino durante las primeras fases de vida del proyecto :

1. Definir los requerimientos : En esta etapa se definen los objetivos y requisitos que debe tener la aplicación. Se consigue mediante un estudio del mercado, intercambio con el cliente,casos de uso o bien cuestionarios a los usuarios para definir sus necesidades y poder implementarlas de la manera más adaptada a la demanda.

2. Definición de las tareas e incrementos: El modelo incremental se basa en varias pequeñas tareas a completar una tras otra para ir construyéndose la aplicación. Una etapa importante viene a ser la definición de dichas tareas en listas que se agruparán según sus objetivos en incrementos .

3. Diseño de los incrementos : Una vez que las tareas y los objetivos del proyecto están claros, conviene definir la evolución del producto en incrementos. Cada incremento tiene un objetivo claro, y en cada incremento se tiene que lograr dicho objetivo antes de pasar al siguiente. Son bloques de códigos funcionales e independientes, y nos aseguramos del correcto funcionamiento de cada uno antes de pasar al siguiente.

4. Desarrollo del incremento : Como su nombre lo indica, en esta etapa se empieza a programar y a desarrollar soluciones para los problemas definidos anteriormente.

5. Validación de los incrementos : Al final de cada iteración, el responsable tiene que aprobar o denegar las soluciones desarrolladas por el equipo de software.
6. Integración de incrementos : Cuando se validan los incrementos, pueden pasar la línea incremental, a la cual cada incremento contribuye y que da, poco a poco, forma al proyecto.
7. Entrega del producto : Cuando todos los incrementos se han validado e integrado, y que se ha comprobado que el producto final cumple con todos los requisitos iniciales, se puede finalmente entregar al cliente, o publicar al mercado.

En la figura 3.1 se puede observar un diagrama de dicho modelo.

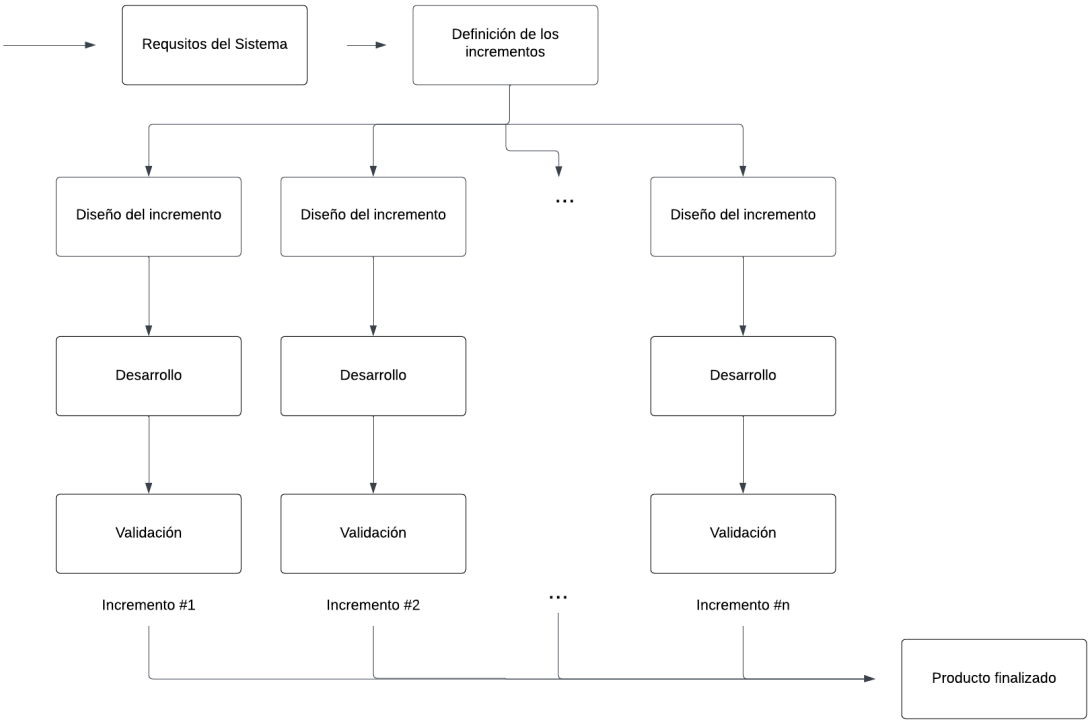

Figura 3.1: Diagrama del modelo Incremental

Existe otra metodología similar al modelo incremental, pero que presenta diferencias claves en su filosofía. Es el modelo Iterativo. La idea principal es la misma, dividir un proyecto grande y complejo en varías tareas simples y fáciles de desarrollar y testear. La diferencia principal viene en la manera de dividir el trabajo. En el modelo incremental, se busca dividir el trabajo en bloques de funcionalidades independientes para poder probar cada una en un entorno controlado, y poder localizar los errores de manera más eficiente, antes de juntar los bloques en el proyecto final. En el modelo iterativo, el desarrollo se hace en ciclos, o iteraciones. En cada iteración, se busca mejorar lo que se ha hecho en las iteraciones anteriores, mejorando el código o añadiendo funcionalidades.

Es un modelo también muy usado, y que es compatible con el modelo incremental. Se puede perfectamente hacer uso de ambos modelos, teniendo una primera división de trabajo basándose en un modelo incremental para la funcionalidades primarias de la aplicación, y luego desarrollar cada una de dichas funcionalidades haciendo uso de un modelo iterativo.

Este tipo de metodologías se usan hoy en día en la mayoría de proyectos software por las distintas ventajas que acabamos de detallar, y por lo tanto es la metodología que

será usada en este proyecto. Primero y como se ha visto, vamos a tener que definir los requisitos de la aplicación :

1. Análisis de requisitos : Es la etapa primordial de cada proyecto siguiendo una metodología incremental. Se determinan los requisitos de la aplicación. Se hará mediante casos de uso y escenarios en el capitulo 4.

2. Análisis conceptual y diseño : Aquí se llevarán a cabo varías subtareas distintas como los diagramas de clase, que nos permitirán identificar los distintos objetos necesarios al funcionamiento correcto de la aplicación, el diseño del modelo de base de datos, y los bocetos de la interfaz de la aplicación.

3. Desarrollo de la solución : Tras modelar las interfaces mediante bocetos y diseñar la base de datos, se desarrollará la solución respetando al máximo los bocetos e intentando cumplir con los requisitos.

4. Validación : Validación del producto mediante pruebas de usuarios y análisis heurística.

En cuanto a incrementos, se ha divido la aplicación en bloques funcionales que se pueden desarrollar por separado y aumentar de gradualmente las funcionalidades de la aplicación.

1. Registro y autenticación : Añadir un sistema de autenticación y una pantalla de creación de perfil y de conexión. La aplicación debe permitir al usuario registrarse y conectarse. Esta cuenta le va a permitir registrar sus aficiones, aficiones que más adelante servirán para la recomendación de sitios.

2. Listado de sitios cercanos : Añadir la funcionalidad de recuperación de sitios cercanos. La aplicación debe poder, basándose en una lista de aficiones, devolver un conjunto de sitios acorde a esta lista.

3. Detalles de sitios y valoración : Añadir una pantalla de detalles y un sistema de valoración. La aplicación puede devolver una pagina de detalles acerca de un sitio con informaciones como el nombre, la distancia o los comentarios. Esta pagina tiene varías funcionalidades. La primera es iniciar el trayecto, que tiene como consecuencia abrir la aplicación GPS preferida del usuario apuntando al sitio elegido. La secunda es permitir al usuario valorar el sitio poniendo una nota, del uno al cinco, y un comentario. Finalmente, el usuario puede añadir o quitar dicho sitio de su lista de sitios favoritos.

4. Sitios guardados e historial : Tener la posibilidad de mantener listas de sitios para los usuarios. Cada usuario tiene asignado dos listas de sitios, una lista de sitios favoritos, o sitios guardados, y un historial de sitios visitados. Las dos pantallas son iguales y tienen las mismas funcionalidades. Se puede abrir la pantalla de detalles desde la lista, y se puede quitar sitios de la lista.

5. Modificación del perfil : Permitir la modificación de perfil. La pantalla de modificación de perfil permite cambiar de contraseña pero sobre todo cambiar las aficiones del usuario, añadiendo nuevas o quitando antiguas.

CAPÍTULO 4
Análisis de Requisitos

En este capitulo, se identificarán y detallarán todas las necesidades que puedan tener los usuarios utilizando la aplicación que se quiere desarrollar. El análisis de requisitos tiene como objetivo reunir y compilar las necesidades tras varios procesos que pueden ser entrevistas, encuestas o reuniones con el equipo de desarrollo. Se tiene que identificar, clasificar y documentar los diferentes requisitos que puede tener la aplicación, así como priorizarlos. Esta etapa permite definir un camino claro para un desarrollo de calidad.

4.1 Captura de requisitos

Este trabajo surge una de necesidad personal, una aplicación para recomendar sitios cercanos y en un intervalo de tiempo corto para utilizaciones esporádicas y espontáneas. De este modo, los requisitos iniciales se hicieron en base a la experiencia de un solo usuario, cuya necesidad de una aplicación dio vida a este proyecto. Sin embargo es una aplicación interesante para cualquier persona interesada en descubrir un barrio o una ciudad nueva.

Como se ha visto en el segundo capitulo de esta memoria, existen muchas aplicaciones cuyo objetivo es la recomendación de sitios, pero ninguna se adapta exactamente a las necesidades. Se necesita una aplicación sencilla que permite recomendar sitios rápidamente. No se trata de planificar eventos, quedadas o viajes, simplemente pasar el tiempo descubriendo un sitio de interés como podría un tiendo o un café. Se tiene que desarrollar una interfaz limpia y minimalista, para no cargar la pantalla con demasiada información ya que la esencia del proyecto reside en una aplicación muy sencilla que tiene como único objetivo la recomendación de sitios de manera eficiente. Por lo tanto se harán todos los esfuerzos para solo mostrar en pantalla información imprescindible a la experiencia del usuario. No sé mostrará información en pantalla si esta información no es importante para el buen funcionamiento de la aplicación, o para ayudar al usuario. Los botones y distintos elementos con los que se podrán interactuar tendrán un nombre explícito, dejando claro su funcionalidad, como “Volver”, “Iniciar Trayecto” o “Valorar” para una navegación intuitiva.

4.2 Requisitos iniciales

El objetivo de la aplicación es permitir a cualquier persona descubrir un barrio o una ciudad a través de actividades de su interés. La aplicación devolverá un listado de distintos sitios. Estos sitios deberán estar a una distancia recorrible en un tiempo calculado a partir del tiempo introducido por el usuario, y que permita disfrutar del sitio. Por ejem-

plo, no se mostrarán sitios accesibles en cuarenta minutos si el usuario solo dispone de cincuenta minutos. Cada sitio aparece junto a la distancia que le separa del usuario. Aparecen también sus tags para poder identificar de qué tipo de sitio se trata.Para poder decidirse, será útil tener la valoración de cada sitio junto a comentarios para que cada usuario pueda contar su experiencia. Si el sitio ha gustado, y se quiere volver, se debe poder añadirlo a una lista de sitios guardados, o sitios favoritos. Además, si se quiere volver a sitios que fueron visitados, deberá de ser posible mostrar un historial. También se da el caso del usuario que ya no está interesado en un tema que sí que indicó estar interesado en un principio. Por lo tanto, se tiene que poder editar posteriormente el perfil. Los requisitos que debe cumplir el sistema son los siguientes :

- Crear una cuenta, registrando sus gustos personales.
- Devolver un listado de sitios acorde a los gustos del usuario y al tiempo disponible.
- Poder ver la nota media de cada sitio antes de darle.
- Poder leer los comentarios dejados por otros usuarios.
- Poder dejar comentarios de cada sitio.
- Poder iniciar el trayecto hacía un sitio.
- Poder añadir un sitio como favorito.
- Poder ver la lista de sitios guardados.
- Poder consultar el historial.
- Poder modificar el perfil de usuario y sus gustos personales

4.3 Casos de Uso y Escenarios

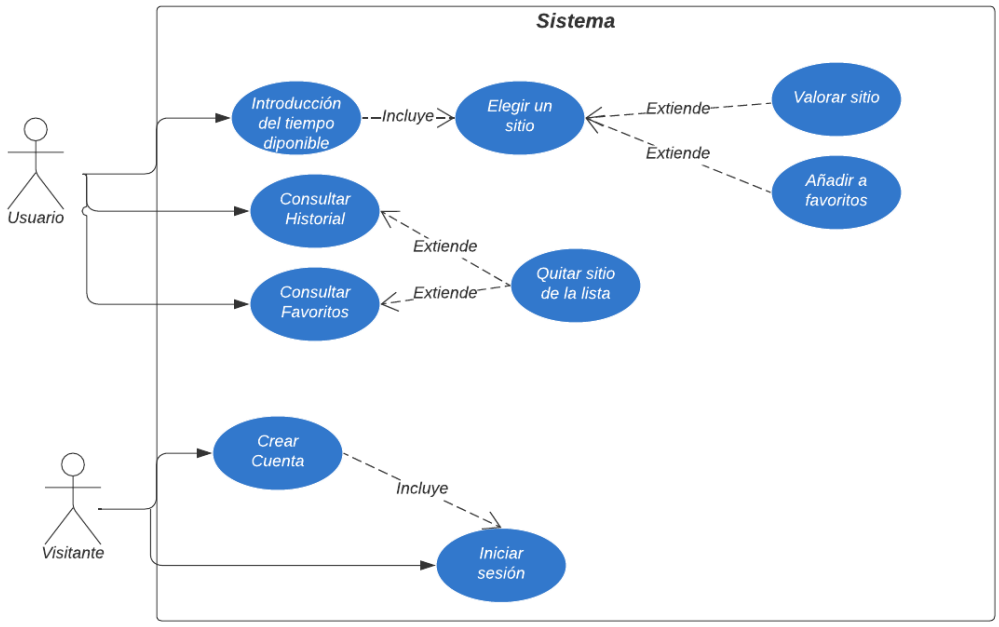

Figura 4.1: Diagrama de casos de uso.

El caso de uso[9] de la figura 4.1 nos muestra distintos ejemplos de uso de la aplicación. La funcionalidad central consiste en iniciar un trayecto. Una vez el sitio fue elegido, el usuario puede iniciar el trayecto. A partir de ahí, podrá volver a consultar dicho sitio en su historial. También podrá valorar el sitio y dejar un comentario, o añadirlo a su lista de sitios favoritos. En esta aplicación sólo existen dos tipos de usuarios, los usuarios registrados, y los nuevos usuarios. Los usuarios registrados disponen de una cuenta y se podrán conectar. Los nuevos usuarios podrán crearse una nueva cuenta. La aplicación devolverá a los usuarios un listado de sitios basándose en sus gustos, su posición y su tiempo disponible. El usuario podrá seleccionar un sitio e iniciar un trayecto. También podrán añadirlo a su lista de sitios favoritos. Finalmente, cada usuario podrá dejar comentarios y valoraciones de los distintos sitios.

Escenario 1 : El usuario crea un perfil y registra sus gustos de forma textual :

Juan desea usar la aplicación y disfrutar de sus funcionalidades. Para ello necesita crearse una cuenta. Juan no es ningún experto en aplicaciones móviles, pero sí que utiliza su smartphone diariamente y disfruta de muchas otras aplicaciones. La creación de una nueva cuenta en esta aplicación es similar a cualquier otra, Juan está familiarizado con la interfaz y puede crear una cuenta sin ningún problema. Introduce su nombre de usuario, su contraseña, una foto, y también se pide unas aficiones. Juan está apasionado de cine, videojuegos y música, procede a introducir, uno a uno, estas tres aficiones.

Escenario 2 : El usuario introduce el tiempo que tiene disponible para descubrir el barrio/sitio :

Juan, por su trabajo, está por unos días en Madrid. Juan es de Valencia. Ha ido a Madrid un par de veces pero al ser una ciudad tan grande, no pretende conocerla, y está en un barrio completamente nuevo. El congreso al que Juan tiene que asistir solo empieza mañana a las nueve, y son las siete de la tarde. Juan tiene delante dos horas antes de ir a cenar. Le gustaría dedicar estas dos horas a alguna actividad que le guste, pero como no conoce la ciudad ni el barrio, no sabe a dónde dirigirse. Juan abre la aplicación, e introduce el tiempo que tiene disponible. Al haber registrado previamente como aficiones la música, los videojuegos y el cine, la aplicación le devuelve un listado de ocho sitios a los que puede llegar en un tiempo razonable para poder disfrutar de la actividad. De estos ocho sitios, puede elegir entre dos salas de conciertos, dos tiendas de videojuegos, tres cines y una sala de juegos arcades.

Escenario 3 : El usuario elige uno de los sitios recomendados por la aplicación :

La aplicación devolvió a Juan ocho sitios. Puede ver el nombre,la distancia ,la valoración de estos sitios para poder elegir uno tanto como los tags que indican a cuál de sus aficiones corresponde cada sitio. Hay una sala de conciertos de Jazz a menos de 400 metros, y Juan es un fanático de Jazz. Procede a pulsar sobre este sitio. Tiene una valoración de 4.7/5, y la mayoría de comentarios reflejan un ambiente increíble y músicos de primer nivel actuando todas las noches. Juan usa el mapa que dispone la aplicación para guiarse hasta llegar a la sala.

Escenario 4 : El usuario valora del 1 al 5 el sitio recomendado, pudiendo dejar algún comentario con el fin de ayudar a otros usuarios. Además decide dejar un comentario para que los futuros usuarios se hagan una mejor idea :

Juan pasó una noche inolvidable, fue un concierto magnífico y, como para agradecérselo a la sala de conciertos, decide dejar una nota máxima y un comentario positivo. En la página de la sala de conciertos en la aplicación, Juan dispone de un botón “Valorar” que le permite dar una nota, y opcionalmente un comentario. Juan deja la nota de 5/5, y un comentario muy positivo describiendo su experiencia.

Escenario 5 : El usuario no se acuerda del sitio que descubrió gracias a la aplicación el mes pasado, lo consulta en el historial :

Seis meses después, Juan está de vuelta a Madrid pero de vacaciones con sus amigos. Quiere enseñarles este sitio de conciertos tan bonito del que no deja de hablar desde que volvió de su último viaje a la capital. El problema es que esta vez está en un sitio bastante más lejos que la última, y no reconoce el barrio, ni podría llegar solo. Abre la aplicación y accede a su historial. Juan utiliza la aplicación muchas veces a la semana, y en seis meses visitó muchos más sitios. Para encontrar la sala de conciertos más rápido, decide usar la barra de búsqueda e introduce el nombre de la sala. Enseguida la vuelve a encontrar. Juan y su grupo de amigos se encuentran a 4 kilómetros de la sala de conciertos, pero según, el concierto vale la pena un viaje un poco más largo, e inicia el trayecto con el botón correspondiente.

Escenario 6 : El usuario desea volver fácilmente a un sitio que le gusto mucho, lo marca como favorito :

Para no volver a perder el sitio, Juan decide guardarlo como favorito. Le da al botón correspondiente. Ahora aparecerá en sus sitios guardados y lo encontrará más fácilmente en nuevas ocasiones.

Escenario 7 : El usuario cambió de aficiones después de un tiempo y quiere registrarlo en la aplicación :

Después de un tiempo, Juan sigue utilizando la app mensualmente. Pero se da cuenta que ya lleva mucho tiempo sin jugar a videojuegos por falta de tiempo y de interés en las últimas novedades. Por lo tanto, ya no le interesan mucho los sitios recomendados por la aplicación que tratan del tema. Abre el apartado “Modificar Perfil”, y en el apartado aficiones, decide quitar el tag “videojuegos”.

CAPÍTULO 5
Análisis Conceptual y Diseño

El análisis conceptual es la fase que sigue al análisis de requisitos en el proceso de desarrollo de software. Se trata de traducir los requisitos previamente establecidos en una estructura lógica. Esta estructura se basa en diferentes modelos y diagramas como el modelo de base de datos, el diagrama de clases, o bocetos de la interfaz.

5.1 Diagrama de clases

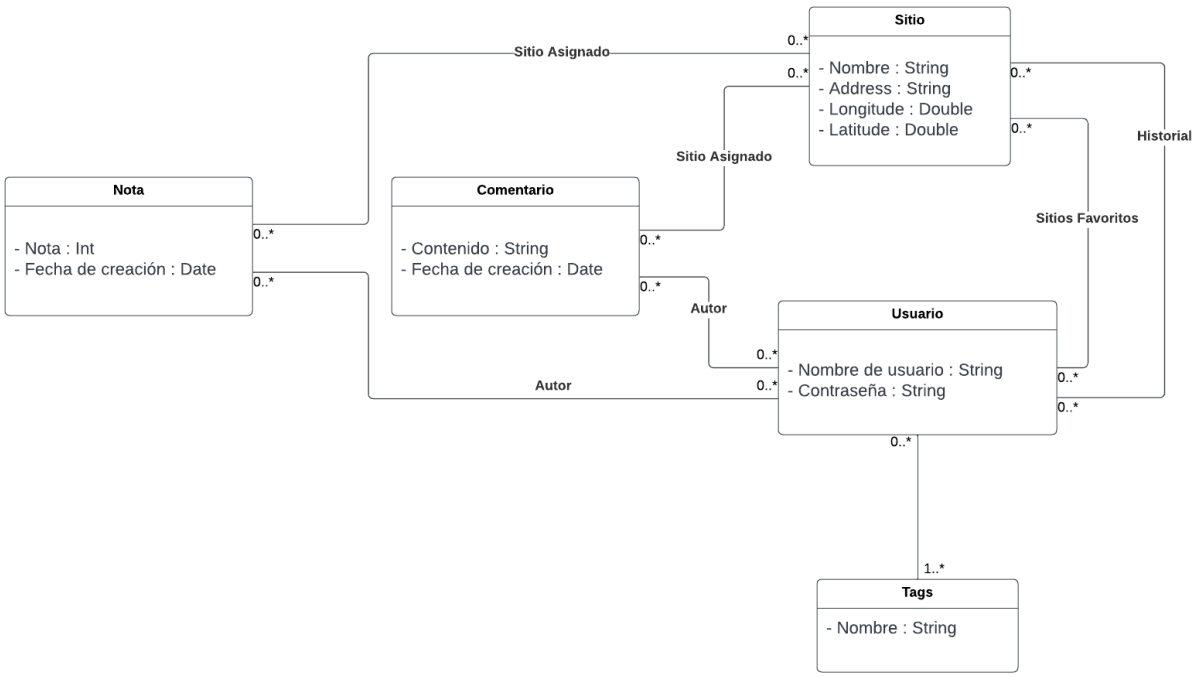

Figura 5.1: Diagrama de clases

En la figura 5.1 se puede observar el diagrama de clases que describe en el sistema a desarrollar en lenguaje UML[1]. Nos permite apreciar una representación de las distintas funcionalidades de la aplicación, tanto como la interactividad de los componentes. Son necesarias cuatro clases al funcionamiento correcto de la aplicación. La primera es la clase Üsuario"que cuenta con varios atributos. Los atributos como "Nombre de usuarioz Çontraseña"son bastante explícitos. También cuenta con una lista de aficiones. Estas aficiones son representadas por una clase "Tags", con un nombre que tiene que ser explicito,

como 'cine' o 'videojuegos'. La siguiente clase es la clase "Sitio"que representa los sitios que puede recomendar la aplicación. Cuentan con un nombre, una dirección y una posición.Un usuario puede tener guardado en sus lista de sitios favoritos varios sitios. Lo mismo pasa con su historial. Para acabar, las ultimas clase son las clases Çomentariosz "Notas". Estas clases tienen como atributos su contenido y su fecha de creación. Los comentarios también tienen un autor que es un usuario y un sitio asignado.

5.2 Modelo de Base de Datos

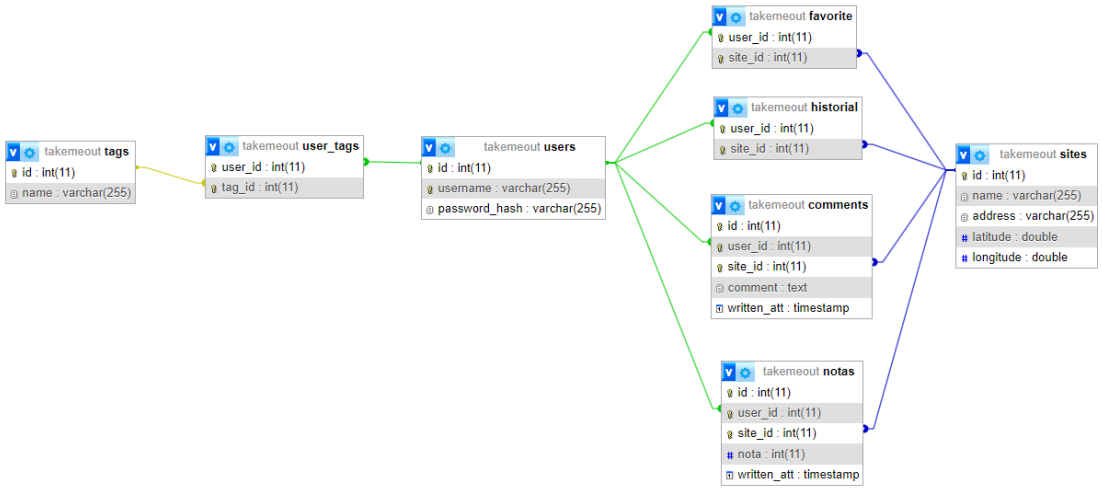

Figura 5.2: Modelo de base de datos

La aplicación utiliza una base de datos relacional SQL. El diseño de la base de datos, aunque puede parecer secundario en un proyecto de software, viene a ser una etapa primordial. Un modelo bien pensada permite facilitar el desarrollo de la aplicación, la transacciones con la dicha base de datos, y permitir interacciones fluidas con el servidor. Por lo tanto, se ha optado por un modelo relacional. Es el modelo más utilizado y más practico cuando se trata de una aplicación que contara con varios objetos interconectados e intercambiando datos en cada momento. En el modelo relacional, cada entidad esta representada por una tabla, con sus características y propiedades representadas en sus columnas. Cada entrada en la tabla representa una iteración de la entidad, como podría ser por ejemplo el usuario 'Juan' en la tabla 'Usuarios'. Cada tabla cuenta con una clave primaría, atributo único cuya función es la identificación de las distintas filas. Para permitir relaciones entre las diferentes tablas, además de las claves primarias, las tablas pueden contener claves ajenas. Las claves ajenas de una tabla son las claves primarias de otra tabla y permiten relacionarla con la misma. Lo podemos ilustrar mediante un ejemplo. Retomando a nuestro usuario 'Juan' de la tabla 'Usuarios'. Dicha tabla tiene como clave primaria el DNI de los usuarios. Otra tabla, la tabla de 'Administrador',que permite relacionar los usuarios que son también administradores, tiene como clave primaria un id generado por la propia tabla, y como clave ajena a la tabla 'Usuarios' el DNI de los usuarios cuya función es la de administrador. El modelo relacional ofrece muchas ventajas en el contexto de este trabajo. Es un modelo muy fácil y sencillo de usar, con gran flexibilidad y escalabilidad si se monta correctamente. También se puede implementar reglas de integridad y restricciones que permiten hacer más eficiente el mantenimiento de los datos y mantener coherencia. El modelo de base de datos de la figura 5.2 es le que ha sido desarrollado para este trabajo y se basa en el diagrama de clase de la figura

5.1 con algunas modificaciones que han facilitado el desarrollo de la aplicación. La tabla 'Users' como su nombre indica, almacena a los usuarios de la aplicación. Tiene tres propiedades, el id, que sirve de clave primaria y permite identificar a todos los usuarios. Un nombre de usuario único y Un hash de la contraseña, que permite identificar los usuarios en el contexto de la aplicación. La tabla 'Sites' guarda los distintos sitios visitados por los usuarios. Esta tabla cuenta con cinco atributos. Un id que permite identificar el sitio, un nombre, una dirección, una latitud y una longitud. Los usuarios y los sitios se pueden relacionar de varías formas. Cada usuario puede guardar un sitio como favorito. Sin embargo cada usuario puede tener varios sitios favoritos, y cada sitio puede ser el favorito de varios usuarios. no se podía relacionar directamente las dos tablas con atributo favorito en la tabla 'Users'. Por lo tanto se ha creado tablas para relacionar los sitios con los usuarios. La tabla favorite tiene solo dos atributos, el 'user_id' y el 'site_id', que relaciona respectivamente el id de un usuario con el id de un sitio. Usa como clave primaría la tupla 'user_id'/'site_id', lo cual permite tener varías entradas con el mismo id de usuario o de sitio, por solo una entrada con el mismo par usuario y sitio. Con la misma lógica, se ha creado la tabla historial, que permite relacionar los usuarios con los sitios que han visitado previamente. La tabla 'Comments' en cuanta a ella, permite registrar los comentarios de los diferentes usuarios sobre los sitios que visitan. Dicha tabla posea cinco atributos, un id único que permite identificar al comentario. Un 'user_id' para identificar al autor del comentario. Un 'site_id' para relacionar el comentario con un sitio. 'Comment' para el contenido del comentario y 'written_att' que permite guardar el momento en el que se escribió el comentario. De la misma manera se ha implementado la tabla 'notas' para guardar las notas que los usuarios dejan a los sitios. Solo se admite un nota y un comentario por usuario por sitio. Por lo tanto, si un usuario intenta dejar una comentario o una nota a un sitio que ya valoró previamente, no se insertara una nueva fila en la tabla, sino que el contenido del comentario o de la nota será actualizado por el nuevo. En el contexto de nuestra aplicación, cada usuario tiene varios aficiones, que denominaremos 'tags'. Cuando un usuario se registra en al aplicación, se le pide sus aficiones. Si es la primera vez que dicha afición esta registrada, se registra en la tabla 'tags' donde se guardan el nombre de las aficiones. Como un mismo usuario puede tener varias, se necesita otra tabla para relacionar las aficiones con los usuarios. Por lo tanto, tabla 'user_tags' permite relacionar un usuario con una afición. Y de la misma manera que ha sido hecho en las tablas 'favorite' e 'historial', esta tabla usa como clavia primaria la tupla 'user_id'/'tag_id'.

5.3 Bocetos de las interfaces de la aplicación

Como se ha mencionado anteriormente, esta aplicación está destinada a un público muy amplio y que no son necesariamente usuarios muy frecuentes de aplicaciones móviles. Por lo tanto, se ha elegido un diseño muy sencillo y reconocible, con funcionalidades que esperable de este tipo de aplicación.

La aplicación funcionará con un principio muy simple, al abrirla, se introducirá el tiempo disponible. Una vez la búsqueda inicializada, un listado de sitios cercanos se pondrá en pantalla en función de los gustos del usuario, de la distancia, y del tiempo disponible. Para usar la aplicación es necesario crearse un perfil, ya que durante la creación, se introducirá las aficiones y los sitios que le interesan al usuario.

A continuación se mostrará un boceto de cada página de la aplicación.

5.3.1. Pagina Principal

La pantalla principal se puede observar en la figura 5.3. Es muy sencilla pero cuenta con un par de funcionalidades interesantes. El elemento principal de esta página permite elegir el tiempo disponible. También cuenta con tres botones. El botón abajo a la derecha permite validar el tiempo elegido e iniciar la búsqueda. En la parte superior hay dos botones, uno, a la izquierda y en forma de corazón para que su función sea más fácil de entender, permite abrir la lista de sitios marcados como favoritos. El segundo, a la derecha, abre un menú drop-down y permite elegir entre tres opciones. Mostrar el historial, modificar el perfil o desconectarse.

Figura 5.3: Boceto de la pagina Principal

5.3.2. Listado de sitios

El boceto de la pagina de listado de sitios se puede observar en la figura 5.4. Una vez dado a iniciar en la página anterior, se mostrará en pantalla un listado de sitios que le podrían gustar por las aficiones y gustos introducidos, y al que puede llegar, y disfrutar, en el tiempo indicado. Cada sitio tiene varias propiedades. La primera su nombre, y el tipo de sitio. Es decir, el tag correspondiente al los del usuario, que detallaremos en la página de “crear una cuenta”. Por ejemplo en esta pantalla se puede ver los tags “biblioteca”, “ramen” y “videojuegos” que han sido registrados previamente por el usuario como parte de sus gustos y/o aficiones. También se puede observar la valoración, sobre 5 del sitio, tanto como la distancia. Una vez elegido el sitio, solo hace falta darle al sitio para poder tener más detalles e iniciar el trayecto. También se puede cancelar la búsqueda y volver a la pantalla anterior dándole al botón “X” abajo a la derecha.

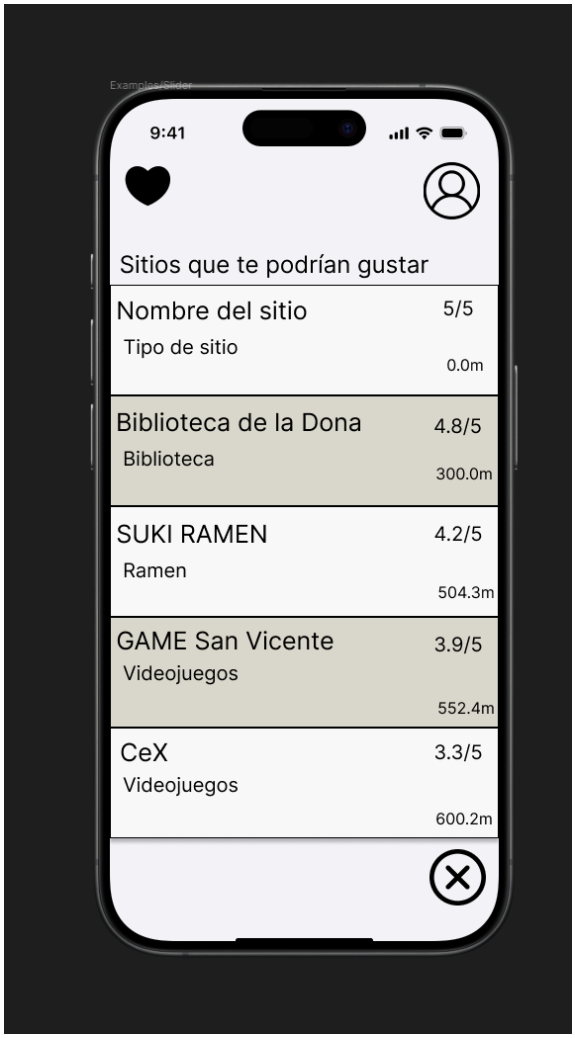

Figura 5.4: Boceto de pantalla de listado de sitios

5.3.3. iniciar un Trayecto

Una vez elegido el sitio y dándole a la casilla correspondiente, se abre un menú con más detalles. Las informaciones básicas siguen en pantalla, como el nombre del sitio, el tipo de sitio, la distancia y la valoración. También se abre en un mapa la localización del sitio, aquí la Universitat Politècnica de València. El botón en forma de corazón sin rellenar permite añadir dicho sitio a la lista de favoritos. El botón "Volver" permite volver a la pestaña anterior, y el de iniciar permite iniciar el trayecto, abriendo la aplicación en google Maps para continuar el trayecto. También se dispone de un botón "Valorar" que permite dejar una nota, del uno al cinco, y un comentario. Abajo del todo se dispone de la lista de comentarios dejados por otros usuarios.Se puede observar en la figura 5.5.

Figura 5.5: Boceto de pantalla de inicio de nuevo trayecto

5.3.4. pantalla de Conexión

Al abrir la página por primera vez, o después de haber cerrado sesión, llegamos a la página de "login", que permite o bien conectarse a una cuenta existente o bien crear una cuenta, gracias al botón de "Crear una cuenta". También se puedo observar en grande el nombre de la aplicación "TakeMeOut". Es una interfaz muy sencilla y similar a muchas aplicaciones del estilo, que es lo que se busca para que el usuario se siente familiarizado con la aplicación sin haberla probado antes. Se muestra en la figura 5.6

Figura 5.6: Inicio de sesión

5.3.5. Nuevo Usuario

Al darle al botón de "Crear una cuenta", podemos crear un nuevo usuario, introduciendo un nombre de usuario, una contraseña, la posibilidad de introducir una foto de perfil dándole al icono, y los aficionados y/o gustos que aparecerán como forma de tag. Este se puede observar en la figura 5.7. La api de google maps funciona con procesamiento de cadenas de texto, por lo tanto, solo falta introducir textualmente lo que se busca, como por ejemplo "Libros" para que devuelva un listado de bibliotecas o librerías. Es basándose en esto que se introducen las aficiones del usuario para devolver un listado de sitios que podrían interesarle.

Figura 5.7: Boceto de pantalla de creación de usuario

5.3.6. Modificar Perfil

En la figura 5.8 se muestra la pantalla donde se puede modificar el perfil de usuario. Es posible cambiar la foto de perfil o añadir, cambiar la contraseña, y modificar el listado de aficiones. En esta pantalla, el usuario llamado Juan tiene actualmente tres aficiones. El cine, los videojuegos, y la música. Si iniciaría una nueva búsqueda, la aplicación le devolverá sitios relacionados con estas aficiones. Puede modificar las, añadiendo nuevas aficiones en el campo correspondiente, o quitando algunas con el símbolo "X" que aparece al lado de cada afición.

Figura 5.8: Boceto de pantalla de Modificación Perfil

5.3.7. Historial

El historial permite ver los sitios visitados. La barra de búsqueda permite buscar un sitio por su nombre. También se puede eliminar un sitio del historial pulsando el botón "X". Si el sitio está guardado como favorito, lo indica el icono de corazón rellenado, si no es el caso, aparecerá el icono de corazón vacío. Se puede añadir o quitar de la lista de favoritos dándole al icono de corazón. El botón volver permite volver a la página anterior.

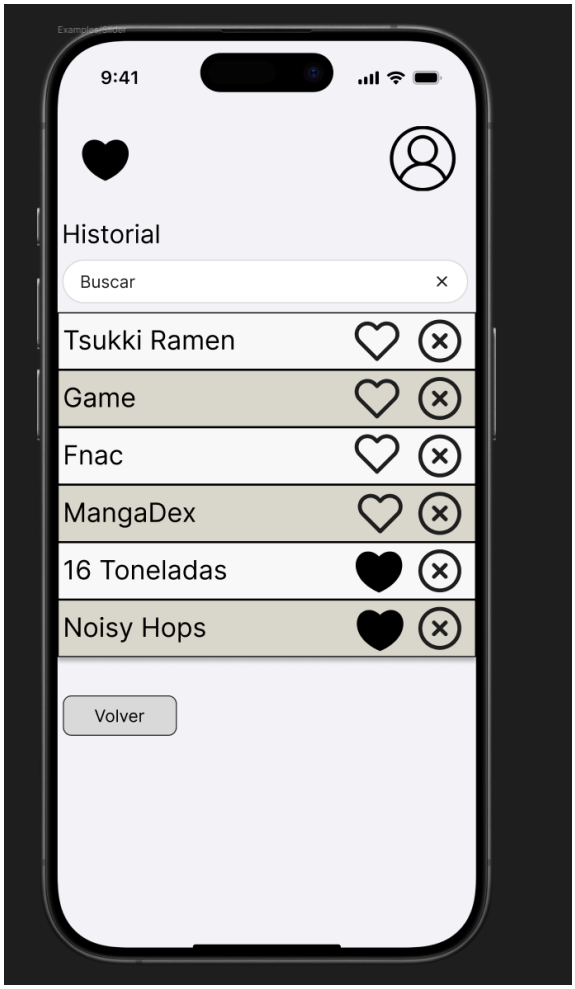

Figura 5.9: Boceto de pantalla de Historial

5.3.8. Sitios favoritos

Con el botón en forma de corazón, accedemos a la lista de sitios guardados como favoritos. La barra de búsqueda permite buscar un sitio específico. Con un simple click en un elemento de la lista, se abre la página detallando el sitio y permitiendo iniciar un trayecto. Con el botón "X", podemos quitar un elemento de la lista. Con el botón "Volver", podemos volver a la pantalla anterior.

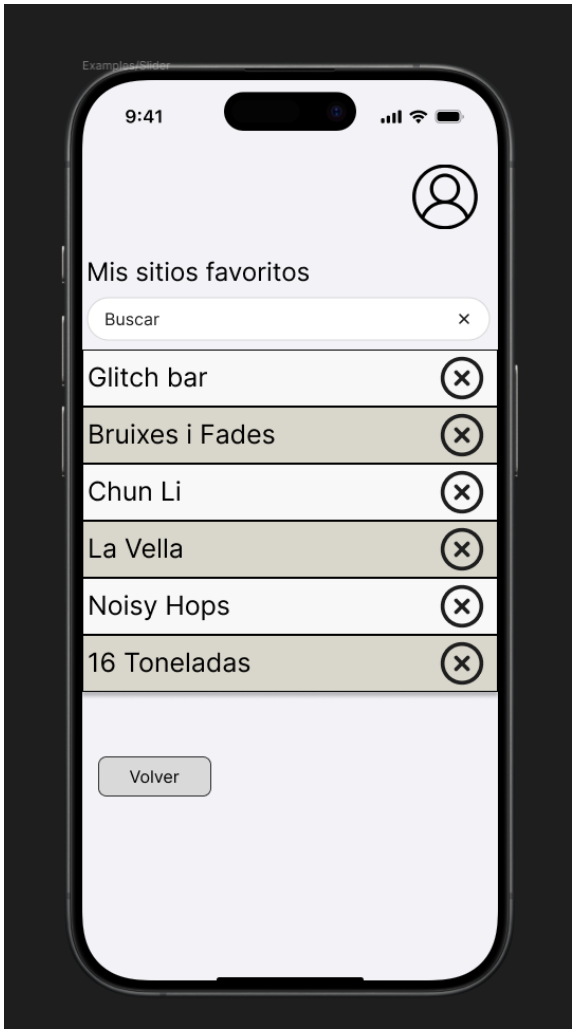

Figura 5.10: Boceto de pantalla de sitios guardados

CAPÍTULO 6
Desarrollo de la solución

En este capitulo se detallarán todas las etapas y tecnologías que fueron necesarias para el desarrollo de la solución, junto con ejemplos de código para poder tener una visión más técnica de la misma.

6.1 Arquitectura

La arquitectura de la aplicación cuenta con tres componentes. La interfaz de la aplicación, o el front-end. El servidor, o back-end, que permite gestionar todas las llamadas a base de datos desde el front-end. Y la base de datos. La aplicación funcione según un modelo de API Representational State Transfer, o REST[4]. Permite la comunicación entre los distintos componentes a través del protocolo HTTP. Este tipo de API es muy utilizado en diversos tipos servicios, dado el hecho de que es muy sencillo de usar, muy escalable y admite varios formatos de datos. Permite recuperar, añadir y actualizar datos de una base de datos mediantes simples peticiones HTTP como GET, POST, PUT o DELETE, y objetos JSON. Una vez las peticiones estén recibidas por el servidor, una función determinada interactuara con la base de datos para llevar a cabo la operación pedida por el usuario desde la interfaz de la aplicación móvil. En el contexto de nuestra aplicación, varías operación interactúan con la base de datos. La aplicación tiene que poder registrar los usuarios y los sitios visitados tanto como permitir el acceso a los usuarios existentes. Cada usuario debe poder registrar sus sitios favoritos, y mantener al día su historial. La aplicación tiene que poder recuperar las aficiones de los usuarios y modificarlas si lo pide el usuario. Finalmente, y para cada sitio, la aplicación tiene que recuperar tanto los comentarios asociados como las notas. Algo importante que hay que tomar en cuenta es que las API REST no mantienen el estado entre las diferentes peticiones. Por la tanto, hay que asegurarse de que cada petición contiene toda la información necesaria. Si, por ejemplo, un usuario esta conectado y pide acceder a su historial, luego sus sitios favoritos y finalmente actualizar sus aficiones, en las tres peticiones que se harán al servidor sera necesario mandar la información del usuario. Aunque que solo se pide recuperar datos de la base de datos, será necesario mandar objetos JSON que permiten identificar al usuario por ejemplo. Por lo tanto, todas las operaciones serán realizadas con la operación POST, que permite mandar datos al servidor. Un ejemplo, sería la figura 6.2 donde, para obtener los sitios favoritos de un usuario, se manda un objeto JSON cuyo contenido es el 'token' identificativo de dicho usuario.

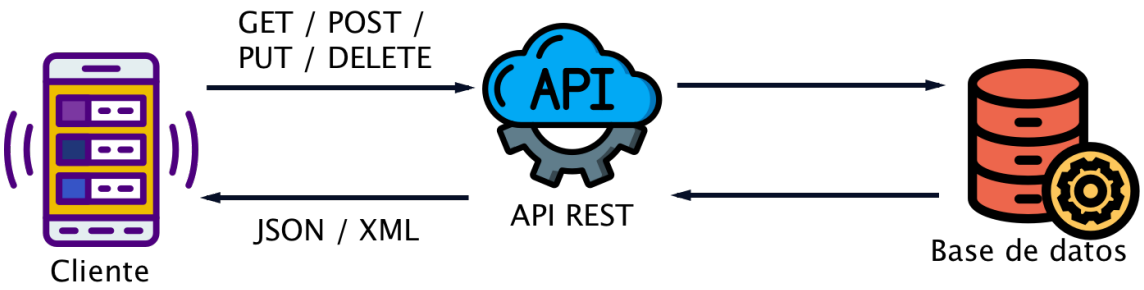

Figura 6.1: API REST

Figura 6.2: Objeto JSON para la obtención de los sitios favoritos de un usuario

6.2 Contexto tecnológico

La aplicación consiste en tres componentes principales que me permiten el correcto funcionamiento de la API REST. Una interfaz movil para las interacciones con el usuario. Un servidor para gestionar las peticiones de la interfaz, mandar peticiones a la base de datos e interpretar sus respuestas, y una base de datos. Para el desarrollo de la interfaz y del servidor, se ha optado por dos frameworks muy actuales y muy utilizados en el desarrollo móvil, React Native para la parte front-end, y Node.js para el back-end. La principal ventaja de usar tecnologías tan actuales es tener a disposición una documentas actualizada y una comunidad muy activa que facilita mucho el desarrollo. Por otro lado, Ambas tecnologías usan Javascript como lenguaje de programación, lo cual hace que pasar del uno al otro es muy sencillo. Finalmente para la base de datos, se ha obtado por MySQL, herramienta muy potente y directa en su funcionamiento para las bases de datos SQL.

6.2.1. React Native

React Native[5] es un framework de desarrollo de aplicaciones móviles híbrido, es decir, permite el desarrollo simultaneo de aplicaciones tanto para IOS como Android. El desarrollo híbrido ofrece varios ventajas frente al desarrollo en tecnologías nativas, pero también presenta algunas desventajas que conviene tener cuenta a la hora de decidirse sobre la tecnología que será usada durante el proyecto. A nivel de rendimiento y aunque React Native o su principal rival, Flutter, han avanzado mucho en los últimos años, no pueden igualar la velocidad de rendimiento de tecnologías nativas cuando los procesos requieren más recursos. En algunas ocasiones, los frameworks híbridos están limitados cuando se tiene que acceder a funcionalidades nativas del dispositivo. Este problema es cada vez menos importante, gracias a la comunidad de desarrolladores implicados en el mantenimiento y la evolución de React Native que lo actualiza con la implementación de nuevas librerías permitiendo el acceso a nuevas funciones y funcionalidades. Sin embargo esta solución tiene un defecto, el tamaño de la aplicación. En efecto, un gran numero de librerías en un proyecto implica un tamaño mayor, y posibles ralentizaciones. Por lo cual es importante importar de las librerías únicamente las funciones que son necesarios

al desarrollo. Sin embargo, el desarrollo híbrido también cuenta con varias ventajas. Primero y ante todo, la principal ventaja de los frameworks híbridos es la posibilidad, como su nombre indica, de desarrollar aplicaciones para varías plataformas con un solo proyecto. El código es el mismo para todos los aparatos, salvo algunas excepciones, y puede ser ejecutado tanto en IOS como Android. Esto permite ahorrar mucho tiempo, y hacer llegar la aplicación a muchos usuarios, además de reducir el coste de desarrollo para una empresa interesada en sacar un aplicación para sus clientes. El desarrollo, mantenimiento y crecimiento de la aplicación se ven facilitados con tecnologías híbridas. Hoy en día se ha vuelto una opción muy viable a largo plazos y muy interesantes para aquellos deseosos en probar el desarrollo de aplicaciones móviles.

React Native utiliza Javascript como lenguaje de programación y la biblioteca React como base. React fue inicialmente pensada como herramienta de creación de aplicaciones web interactivas desarrollada por Meta (antiguamente Facebook). React funciona con componentes reutilizables. Cada componente es una pieza de la interfaz, como en un puzzle. Estos componentes pueden ir de una pantalla completa a un simple botón. Este modo de funcionamiento permite reducir mucho la cantidad de lineas de código. Por ejemplo si diez pantallas requieren un botón 'Volver', solo hará falta programarlo una vez, exportarlo y luego llamarlo en cada pagina que lo necesite. Cada uno de estos componentes tiene un estado y unas propiedades. En el caso del botón, puede estar presionado, o no. Y en lugar de describir manualmente cómo cambia la interfaz frente a los cambios de estado, simplemente hace falta describir como debería verse la aplicación en función del estado actual de los datos. Esto se llama renderizado declarativo. React Native se basa en React para construir sus interfaces, traduciendo los componentes React en componentes nativos para IOS y Android. React Native usa un flujo de datos unidireccional, los datos fluyen de los componentes padre hacia los componentes hijos, permitiendo un desarrollo más agradable a la hora de depurar el código. Para comunicar datos, el padre manda información vía las propiedades, o props, del hijo. De esta manera, aunque el componente se utiliza varías veces, puede tener estados distintos en función de su configuración por su componente padre. Se da un ejemplo en el trozo de código siguiente, donde el componente hijo devuelve en pantalla el contenido de la propiedad 'titulo', pasada por el componente padre.

```
1   const hijo = ({titulo}) => {
2     return <Text>{titulo}</Text>
3 };
4
5 const padre1 = () => {
6     return <hijo titulo = 'Hijo 1' />
7 };
8
9 const padre2 = () => {
10    return <hijo titulo = 'Hijo 2' />
11 }
```

6.2.2. Librerías externas

React Native es una herramienta de código abierta con una comunidad masiva y dedicada a mejóralo. Por lo tanto existen muchas librerías externas que nos van a permitir hacer uso de funcionalidades que no están implementadas directamente en React Native, pero que fueron añadidas para mejorar la experiencia. En este trabajo necesitaremos principalmente cuatro librerías :

1 - React-Navigation[6] : Es una librería indispensable de enrutamiento y navegación diseñada para aplicaciones móviles. Permite tener varías pantallas, y pasar datos entre

ellas. De esta forma, no dispondremos de limite en el flujo de datos, ya que, al tener varías pantallas, será interesante pasar de datos entre las diferentes pantallas.

2 - react-native-get-location : Como su nombre indica, está librería es imprescindible en el contexto de este proyecto, ya que es librería encargada de recuperar la posición del usuario en tiempo real. Es muy fácil de usar, y obtener la posición es cuestión de un par de lineas de código, como lo muestra la figura 6.3.

```
import GetLocation from 'react-native-get-location'

GetLocation.getCurrentPosition({
    enableHighAccuracy: true,
    timeout: 60000,
})
.then(location => {
    console.log(location);
})
.catch(error => {
    const { code, message } = error;
    console.warn(code, message);
})
```

Figura 6.3: Ejemplo de codigo para obtener la posición del usuario

Es compatible con IOS y Android, con precisión configurable para aligerar la carga o aumentar la precisión de los resultados y funciona con promesas. Es decir, está basado en un funcionamiento asíncrono. La promesa representa una posible respuesta si la petición devuelve un resultado correcto, o un error, y, en cuando llegue esta respuesta, será tratada. Permite no bloquear el funcionamiento de la aplicación en la espera del resultado, pero es importante usar la palabra clave await en el código, para esperar a la respuesta antes de seguir, si la respuesta es necesaria al funcionamiento de la función.

3 - Axios : Es una librería de Javascript que permite gestionar peticiones HTTP. En este proyecto, será necesario generar peticiones http para obtener el listado de sitios cercanos a traves de la API de Google Maps, y para las interacciones con el servidor vía la API REST. También es compatible con las promesas y la sintaxis '.then() / .catch()' o 'async/await/.

4 - react-native-maps : Finalmente la biblioteca 'maps' permite integrar mapas en la aplicación, lo cual será necesario para una mejor experiencia de usuario.

6.2.3. Node.js

Node.js es famoso para ser un entorno de ejecución de Javascript. Cada persona deseando ejecutar código javascript tiene que usar un interpretador, ya que no es un lenguaje compilado, sino interpretador. El interpretador más famoso de javascript es V8, el motor de chrome, y Node.js esta basado en V8. Permite ejecutar código del lado del servidor, y crear un entorno de desarrollo completo en un solo computador. Es un framework de back-end, que se encarga de toda la lógica oculta al usuario para permitir la mejor experiencia posible. Node.js tiene varías características que lo hace muy interesante y muy competitivo. Primero, el motor V8 de Chrome en el que se basa Node.js convierte el código Javascript en código máquina nativo, lo cual permite una ejecución muy rápida. Node.js utiliza un sistema asíncrono orientado a eventos, lo cual permite manejar simultáneamente múltiples solicitudes de manera concurrente sin bloquear la ejecución del código. Es muy útil en el contexto de una aplicación móvil con varios usuarios que

pueden estar conectados a la vez. También lo hace muy escalable. Finalmente, al usar Javascript como lenguaje, permite facilitar y reducir la carga de trabajo entre el front-end y el back-end. Al tener que usar solo un lenguaje de programación entre las dos, la lógica del lenguaje y su complejidad es la misma. En el contexto de este proyecto, solo se necesitaba un servidor para gestionar las distintas peticiones a la base de datos, para recuperar, guardar, o modificar información vía peticiones HTTP. En el contexto de una API REST Node.js es una de la mejores opciones dado todos los puntos vistos antes.

6.2.4. MySQL

Para la base de datos, se ha optado por la opción más sencilla y más adaptada al contexto del trabajo, y también una tecnología ya familiar dado que se ha visto en varías asignaturas de la carrera de ingeniería informática. MySQL[8] es un sistema de bases de datos relacional, que utiliza el lenguaje SQL, o Structured Query Lenguaje. En SQL las peticiones se hacen en forma de consultas, o queries siguiendo un lenguaje lógico. Por ejemplo , en una base de datos de estudiantes, con propiedades como nombre y nota media,para obtener el nombre de todos los alumnos cuya nota es superior a 8, la consulta es la siguiente :

```
1 SELECT nombre FROM alumnos
2 WHERE nota > 8;
```

Además de ser un lenguaje muy sencillo, también es potente, y permite la implementación de reglas adicionales para mantener coherencia en los datos. Por ejemplo no tener nombres de usuarios repetidos, o solo permitir una sola tupla de datos en tablas como la tabla de sitios favoritos o del historial. MySQL almacena tablas relacionales, cada tabla representado un objeto del sistema o una relación. Todo el sistema se basa en claves primarías y ajenas para manejar estructuras de datos y relaciones, como lo hemos detallado en la sección 5.2.En nuestro trabajo, estas tablas son las tablas representadas en la figura 5.2. Su simplicidad de uso hace de MySQL una de las mejores opciones para base de datos sencillas. Además la optimización de MySQL permite realizar consultas rápidas para permitir al usuario no estar esperando y mejorar su experiencia. Otro punto a favor de MySQL es su escalabilidad, pudiendo soportar grandes volúmenes de datos. En el contexto de una aplicación como esta en un entorno real, es muy importante tener bases de datos escalables, ya que nunca se sabe cuantos usuarios tendrá la aplicación. MySQL tiene más ventajas en entornos más profesionales como su compatibilidad en la nube o la replicación de datos, pero no son interesante en el contexto de este trabajo.

6.2.5. Google Maps API

Nuestra aplicación permite recomendar sitios cercanos a la posición del usuario. Lanzar un satélite no es una opción viable por razones obvias, por lo tanto se ha optado por una opción más económica y sencilla. La API de Google Maps permite obtener información sobre muchos establecimientos cercanos. La aplicación Google Maps se puede buscar sitios por su tipo, como por ejemplo en la figura 6.4 donde la aplicación devuelve un conjunto de sitios correspondientes a la consulta 'Tienda de música'. De la misma manera, se puede obtener la API para obtener sitios cercanos a la posición del usuario, usando sus preferencias previamente guardas para generar consultas. Concretamente la API places permite obtener información sobre una gran variedad de sitios cercanos a una posición, lo cual es exactamente lo que se busca en este trabajo. Para poder usar la API, solo hace falta registrase y pedir una llave que ira asociada a cada petición. En un entorno local de desarrollo, su uso es completamente gratis.

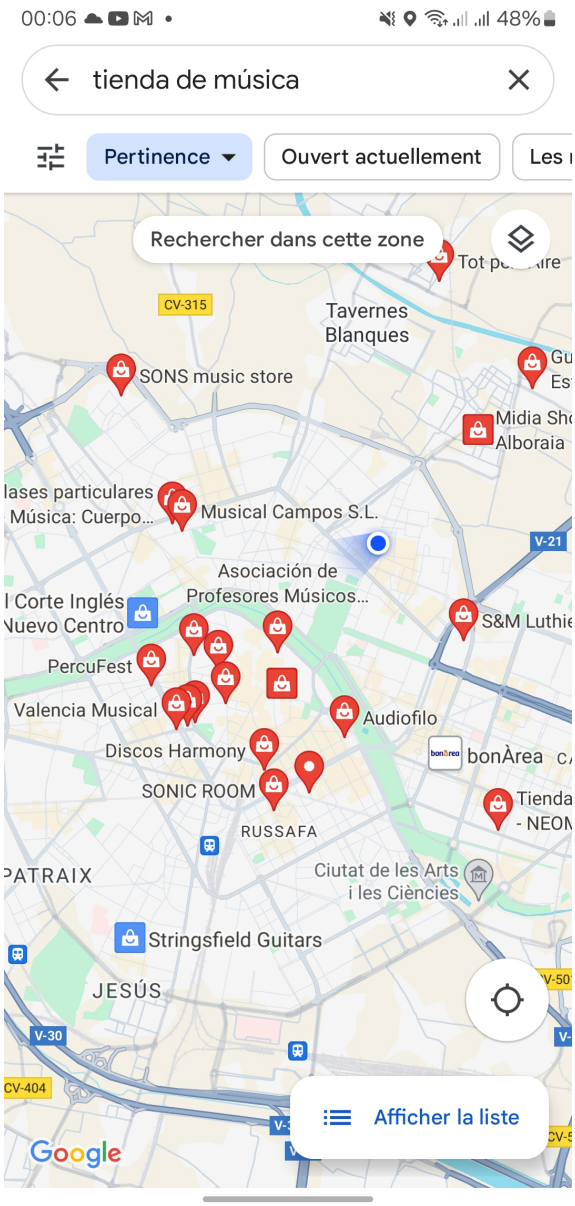

Figura 6.4: Ejemplo de petición de 'Tiendas de música'

6.2.6. Postman

Postman es una herramienta de desarrollo de API que permite facilitar el desarrollo y diseño mediante pruebas de peticiones. En el contexto de este trabajo, permite hacer peticiones al servidor para probarlas y comprobar su funcionamiento antes hacerlas directamente a partir de la interfaz de la aplicación. Permite probar cada petición por separado y localizar los problemas cuando una petición falla, o no tiene el comportamiento esperado. Cuenta con una interfaz gráfica que se puede observar en la figura 6.5. Permite elegir el tipo de petición, como POST, PUT o DELETE, y el cuerpo de la petición con objetos JSON. Ha sido muy útil durante el proceso de depuración ya que permitió aislar los diferentes componentes de una transacción, y localizar el error de forma rápida y sencilla.

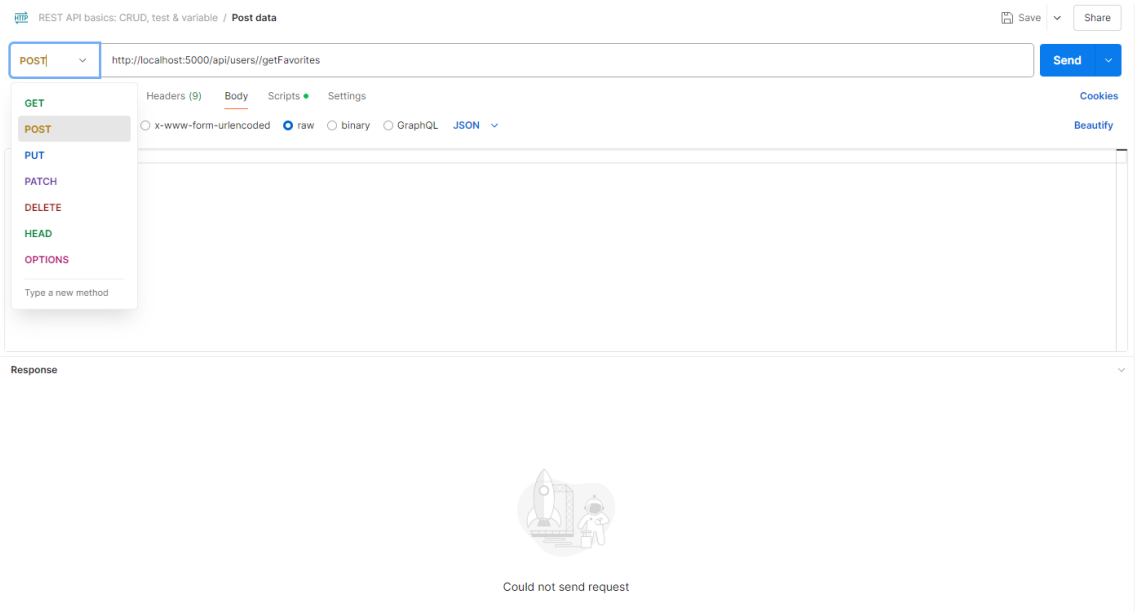

Figura 6.5: POSTMAN : Interfaz gráfica

6.3 Ejemplos de código

En este capitulo se detallará de manera más técnica el desarrollo de la aplicación, los problemas que se encontraron y como se solucionó, con ejemplos de código extraído del producto final.

6.3.1. Interfaz

La interfaz fue completamente desarrollada en Javascript con el framework React Native. En React Native, tanto el estilo como el código y las funciones se encuentran en el mismo archivos. Por motivos de legibilidad del código y de depuración, se suele programar cada componente a parte, para luego juntar los distintos elementos en un archivo llamado 'index.js'. De esta manera, los componentes pueden ser reutilizables por otra paginas sin la necesidad de volver a programarlos. Por ejemplo, el header que aparece en la mayoría de las pantallas solo se programó una vez para luego ser llamado en el index de la pantalla que lo requiere. Para demostrar esto, en la figura 6.6 aparecen los archivos de la pantalla denominada 'Listado' encargada de devolver el listado de sitios cercanos. El código del archivo index.js aparece a continuación :

```
1  import React from 'react';
2  import Listado from './listado';
3  import Header from '../header';
4  export default function Principal(){
5
6
7      return(
8          <>
9          <Header />
10         <Listado/>
11         </>
12     )
13 }
```

Los componentes Header y Listado se importan y la función se encarga de devolver el resultado los componentes hijos.

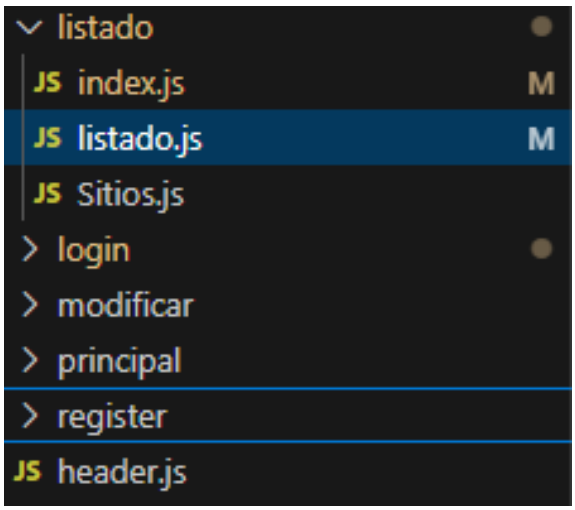

Figura 6.6: Arquitectura de archivos de la pantalla 'Listado'

Cada componente es en realidad una función, que suele ser exportada para poder usarse en un componente padre. Toma como argumento los datos que necesita del componente padre para ser usado aquí, o pasarlos a otro componente hijo del mismo. La función se divide en dos partes. Primero la parte lógica, en la que se inicializan todas las funciones, variables y llamadas que son necesarias al funcionamiento lógico del componente, y el 'return', que se encarga de devolver los elementos visuales de la interfaz. Y para acabar, se puede definir un estilo para el componente gracias a una función de React Native, Stylesheet, que permite definir estilo para los componentes. Un buen ejemplo es para ilustrar esto es el componente 'Aficion' de la pantalla de nuevo usuario de la figura 5.7. En esta pantalla el usuario introduce sus aficiones y aparecen en pantalla en forma de burbujas. Este componente trata de definir estás burbujas con el código siguiente :

```
1  export default function Aficion({tag,deleteTag}){
2
3      const onPressDelete = () =>{
4          deleteTag(tag)
5      }
6
7      return(
8          <TouchableOpacity style={styles.btn} onPress={onPressDelete}>
9              <Text style={styles.txt_btn}>{tag}</Text>
10             <Image source={require('../../icons/cross.png')} style={styles.
                   x}/>
11         </TouchableOpacity>
12     )
13 }
14 const styles = StyleSheet.create({
15     btn:{
16         backgroundColor:'#D9D9D9',
17         borderWidth:1,
18         flexDirection:'row',
19         alignSelf:'flex-start',
20         alignItems:'center',
21         borderRadius:20,
22         marginLeft :5
23     },
24     txt_btn:{
25         color:'black',
```

```
        fontSize:18,
        marginLeft:10,
        marginTop:3,
        marginBottom:5
    },
    x:{
        height:15,
        width:15,
        marginRight:10,
        marginLeft:7
    }
})
```

El componente afición necesita dos objetos del componente padre, el tag, que es el nombre de la afición, y una función 'deleteTag' que permite eliminar una afición. Esta función no puede declararse directamente aquí, porque el componente 'Aficion' no se encarga de devolver todas las aficiones del usuario, este es el trabajo de su componente padre. Sin embargo, el botón que llama a esta función esta situada en este componente, por la tanto, se define una función 'onPressDelete' que ejecuta la función 'deleteTag' definida en el componente padre. Dentro del 'return' se puede observar como se construye, con un '<TouchableOpacity/>, elemento interactivo de React Native que deja más libertad de personalización que un botón. Dentro un '<Text>', elemento textual, para poder devolver en nombre de la afición pasada por argumentos desde el padre, y una imagen para el icono. Este componente esta llamado en dos sitios, en la pantalla de crear perfil y en la pantalla de modificar el perfil. Las dos llamadas se hacen de la misma manera, y se muestra el código siguiente del componente 'Form':

```
import React, { useState } from "react";

                ...

export default function Form() {
    const [tags,setTags] = useState([]);
    const [tagName, setTagName] = useState('');

                ...

        const addTag = () => {

        if (tags.length > 5) {
            Alert.alert('Solo se admiten un maximo de 5 aficiones.');
            return;
        }

        if (tags.some(t => t === tagName)) {
            Alert.alert('Las aficiones no se pueden repetir');
            return;
        }

        if (tagName.trim()) {
            setTags([...tags, tagName]);
            setTagName('');
        }

    }
    const deleteTag = (tag) => {
        let newTags = []
        tags.forEach(t => {
            if(t !== tag){
                newTags.push(t)
                return
            }
```

```
36          })
37          setTags(newTags)
38      }
39
40              ...
41
42      return(
43          <View>
44                  ...
45
46              <TextInput value={tagName} onChangeText={onChangeText}
                    onSubmitEditing={addTag} blurOnSubmit={false}  style={
                    styles.input} />
47              <View style={styles.tagContainer}>
48                  {tags.map((tag) => {
49                      return <Aficion tag = {tag} deleteTag={deleteTag}/>
50                  })}
51              </View>
52
53                  ...
54
55          </View>
56      )
57  }
```

Algo que no se ha comentado antes de lo componentes de React Native y que conviene explicar para entender mejor este código es el estado. El estado de un componente es un objeto que contiene toda la información dinámica de un componente que influye en su renderización, es decir, todos los datos que pueden cambiar durante el ciclo de vida de dicho componente. En el caso de este componente, hay dos datos que pueden cambiar durante su ciclo de vida, la lista de aficiones, ya que se pueden añadir y quitar, y el nombre de la afición que se esta añadiendo y pasando al componente hijo 'Aficion'. Este estado es privado y local al componente. Al hora de modificar el estado, hay que tener cuidado, no se puede hacer de cualquier manera. Como afecta directamente a la visualización de ciertos elementos del componente, podría no actualizarse correctamente, tener problemas de inconsistencia de datos o directamente causar el crash de la aplicación. Para evitar eso, React Native proporciona una herramienta para manejar el estado en toda seguridad, la función 'useState()'. Es función permite definir una variable de estado, una función para modificarla y valores por defecto. En nuestro ejemplo, usamos el 'useState()' en 'const [tags,setTags] = useState([]);' en la linea 3 para definir el array 'tags' que contendrá todas las aficiones. La función 'setTags' permite sobrescribir el contenido de 'tags' y se inicializa como array vació '[]'. Es lo mismo para la variable 'tagName' en la linea 4 para determinar el nombre de la afición a añadir. Más abajo se definen las funciones 'addTag' y 'deleteTag' que permiten respectivamente añadir y quitar una afición. Hacen uso de las funciones 'setTags' y 'setTagName' para modificar el contenido del array 'tags' la variable 'tagName'. Gracias a esto, el estado se actualiza de manera segura. Hay dos componentes interesantes más en este código. Primero el '<TextInput/>' de la linea 42. Gracias a que React Native permite hacer uso de funciones nativas del dispositivo podemos acceder a interrupciones de teclado. En concreto, usamos la tecla 'Enter' del teclado para poder añadir una nueva afición, gracias al método 'SubmitEditing' del componente '<TextInput/>'. Y por ultimo, entre las lineas 43 y 47, podemos observar como la aplicación devuelve el listado de aficiones, gracias a la función 'map', que permite ejecutar una misma función sobre todos los elementos de un array, aquí el array 'tags'. Por cada elemento, se devuelve una nueva instancia del componente hijo 'Aficion'. Estos dos elementos, 'Aficion' y 'Form' son muy representativos del desarrollo de interfaz de React Native y el resto de componentes, a diferencia de su lógica, funcionan igual.

Para acabar sobre la interfaz, hace falta explicar el funcionamiento de la navegación en el proyecto. Se ha mencionado el uso de librería React-Navigation para ello. Es un librería muy potente que cuento con varios tipos de navegación y un alto nivel de personalización. En este proyecto se ha optado por una navegación en 'Stack' o en pila. En este modo, la paginas se ponen una 'encima' de la otra cada vez que se abre una nueva, y permite guardar una trazabilidad de las pantallas visitados y y su orden para poder volver atrás de manera sencilla. Su implementación se hace vía un archivo 'Navigation.js' en la raíz del proyecto. Tiene que importar el componente padre index.js de todas las pantallas para implementar la navegación. En el código siguiente, se puede observar como, en la función 'Mystack', la definición de las distintas pantallas :

```
import {NavigationContainer} from "@react-navigation/native";

            ...

import Listado from './src/pantallas/listado';

            ...

const Stack = createStackNavigator();

function MyStack(){
    return(
        <Stack.Navigator
            initialRouteName= 'Login'
            options={{
                headerShown: false
            }}

        >

                ...

            <Stack.Screen name = 'Listado' component = {Listado}
            options={{
                headerShown: false
            }}/>

                ...

        </Stack.Navigator>
    )
}
export default function Navigation(){
    return(
        <NavigationContainer>
            <MyStack/>
        </NavigationContainer>
    )
}
```

Cada pantalla esta representada por un '<Stack .Screen/>' con un nombre, y el componente que le corresponde, además de algunas opciones de personalización. Todas las pantallas se encuentran dentro de un '<Stack.navigator/>' que da la opción de definir una pantalla inicial. Para navegar entre las distantas pantallas, no es más complicado que usar la función useNavigation().navigate('Nombre de la pantalla'). Además del nombre de la pantalla a visitar, 'navigate()' admite como otros parámetros objetos que se pueden pasar a la otra pantalla. De esta manera, las distintas pantallas pueden pasarse datos entre ellas. Esto se ilustra en la pagina principal de la aplicación. Además de devolver sitios según los gustos del usuario, la aplicación también tiene en cuenta el tiempo disponible

del usuario, que introdujo al entrar en la aplicación. La pantalla principal tiene que pasar esta información a la pantalla siguiente, la pantalla donde se devuelve el listado de sitios. El funcionamiento se demuestra en el código siguiente :

```
//Pagina principal :
            ...
export default function Counter(){
    const navigation = useNavigation()
            ...
    const goToLista = async () =>{
                ...
            var time = parseInt(h) * 60 + parseInt(min)
            navigation.navigate('Listado', {time,tags : newTags});
                ...
    }
    return(
        <>
                ...
        <TouchableOpacity onPress={goToLista}>
            <Image
            style={styles.checkMark}
            source={require('../icons/check-mark.png')}
            />
        </TouchableOpacity>
    </>
    )
}

//Pagina de listado :
        ...
export default function Lista() {
        ...
    const route = useRoute()
    var time = route.params.time
        ...

}
```

Tras definir la variable 'time' en la pagina principal, la pasa al componente 'Listado' para que puede devolver sitios cercanos en función del tiempo disponible. La pagina de listado a su vez recupera la variable mediante la función 'useRoute()'.

6.3.2. Recuperación de sitios de interés

Esta aplicación tiene como objetivo la recomendación de sitios cercanos, basándose en los gustos del usuario y su tiempo disponible. Por lo tanto, unas funcionalidades claves son la recuperación de la posición del usuario y la recuperación de sitios cercanos. Empezando por la posición en tiempo real del usuario, recuperar su posición es fundamental, y se hace de manera sencilla gracias la librería react-native-get-location. Como se ha visto anteriormente, recuperar la posición del usuario se hace en un par de lineas de código, pero trae un problema. Recuperar la posición del usuario es una operación lenta y asíncrona, pero necesaria. Además, varias pantallas la necesitan para su correcto funcionamiento. Las paginas de 'listado', 'favoritos' y el historial, necesitan obtener la posición del usuario para devolver la distancia hasta los sitios. Y dado que la operación de recuperación de la posición tarda entre uno y tres segundos según los casos, no es viable esperar tanto entre cada petición. Sería un problema para la experiencia del usuario. Por la tanto, la solución obvia es pedir la posición una vez, durante la apertura de la aplicación, y luego pasar las coordenadas a las paginas que las necesitan. Pero pasar los datos a todas las paginas hasta esperar que una las necesite no es la solución la más óptima,

por lo tanto, se he hecho uso de un método de React hecho para este tipo de situaciones, el 'AsyncStorage' o tienda asíncrona. Permite de manera sencilla almacenar datos de manera persistente datos del tipo clave-valor. Es una solución local para manejar datos en todo el contexto de la aplicación sin tener que pasarlo a cada componente. El valor se inicializa con 'setItem('calve' : valor)' y se recupera con 'getItem('clave') desde cualquier punto en la aplicación. Al ser asíncrono, se necesita tener un poco más de cuidado para asegurarse de que los datos se han pasado. Sin embargo no está diseñado para almacenar volúmenes de datos excesivamente altos, y solo puede almacenar cadenas de caracteres. Esta manera de guardar y pasar los datos es perfecta para este caso, ya que solo hace falta guardar la longitud y latitud del usuario cuando carga la pantalla principal, y luego pedir estos datos cuando se necesitan. Se propone analizar el código del archivo index.js de la pantalla principal para entender el funcionamiento de la recuperación de la posición además del almacenamiento de dicha posición :

```
            ...

export default function Principal(){
    const[loading,setLoading] = useState(true)
    useEffect(()=>{

        const setPos = async () => {
            try{
            const location = await GetLocation.getCurrentPosition({
              enableHighAccuracy: true,
              timeout: 60000,
            });
                return location
            }catch(error) {
              console.error('error : ' + error)
              setLoading(false);
              return null;
            }
          }
          const showMustGoOn = async () => {
            const loc = await setPos();

            if (loc) {
                AsyncStorage.setItem('latitude',loc.latitude.toString());
                AsyncStorage.setItem('longitude',loc.longitude.toString());
                setLoading(false)
            } else {
                console.log("No location available");
                setLoading(false);
            }
        };
        showMustGoOn();
    },[])

    if(loading) return <Text>Loading....</Text>
    return(
        <>
        <Header/>
        <Counter/>
        </>
    )
}
```

La primera funciona que hay que detallar es la función 'useEffect()' que no se ha explicado aún. Este método permite realizar operaciones que no entran directamente en el proceso de renderización de la aplicación. En nuestro caso, lo utilizaremos para ejecutar peticiones al servidor, o recuperar datos como la posición del usuario en este ejemplo,

antes de montar el componente. La función 'setPos()' en la linea 5 permite recuperar la longitud y la latitud del usuario. Al ser un método asíncrono, hay que tomar precauciones a la hora de inicializar las variables. Por lo tanto, en vez de llamar directamente a la función 'getPos()', se crea otra función, también asíncrona, llamada 'showMustGoOn()' cuyo objetivo es inicializar correctamente las variables. De esta manera, la variable 'loc' en la linea 21 espera a que se haya obtenido las posiciones, con la palabra clave 'await' que permite esperar a la respuesta de la función antes de seguir con el resto de operaciones. Una vez obtenida la posición, se inicializan las variables 'latitude' y 'longitude' con los valores obtenidos. Como ultima garantía de que el código no se ejecutará antes de tener los datos correctamente inicializados, una variable 'loading', inicializada a 'true' permite esperar antes de devolver el verdadero contenido de la pagina. El contenido cambia en el momento en el que se ha recibido una respuesta para la posición, y 'loading' se pone a false, permitiendo la redenrización de la pagina. Para poder ejecutartodo correctamente, primero hace falta modificar algunos archivos internos del proyecto para poder permitir a la aplicación preguntar por la autorización de usar los datos de navegación. Sino, no se podrá acceder a la posición. Eso se hace añadiendo las dos lineas siguiente en el archivo de configuración de Android AdnroidManifes.xml :

```
1 <!-- Define ACCESS_FINE_LOCATION if you will use enableHighAccuracy=true
      -->
2 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
3
4 <!-- Define ACCESS_COARSE_LOCATION if you will use enableHighAccuracy=false
       -->
5 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Y las dos lineas siguientes en el archivo de configuración de IOS Info.plist :

```
1 <key>NSLocationWhenInUseUsageDescription</key>
2 <string>This app needs to get your location...</string>
```

Ahora que la variables de longitud y latitud han sido correctamente inicializadas, se pueden recuperar los sitios cercanos. En la pagina principal, antes de pasar a la pagina siguiente, se hace una llamada al servidor para recuperar las aficiones del usuario conectado gracias a este código :

```
1                 ...
2 const token = await AsyncStorage.getItem('token');
3 const resp = await axios.post("http://192.168.18.12:5000/api/users/
      getAficiones",{token})
4 const newTags = (resp).data.map(item => item.tag_name)
5 setTags(newTags)
6 navigation.navigate('Listado', {time,tags : newTags});
```

La variable 'token' consiste en un identificador del usuario. Detallaremos su funcionamiento más adelante. Por ahora, permite al servidor saber de que usuario se trata. Lo que interesa en este código es la petición al servidor, que devuelve como resultado las aficiones del usuario. Una vez obtenidas, están pasadas, junto con el tiempo disponible introducido por el usuario, a la pagina de listado. Esta pagina a su vez recupera primero todos los datos que necesita para construir una petición a la API de Google Maps. Este proceso se detalla a continuación :

```
1             ...
2     var time = route.params.time
3     const apiKey = 'AIzaSyDLJzWNXb7EPryuohfAVCiBX6NmCJrJnjQ';
4     const radius = (85 * time)/3;
5     const tags = route.params.tags;
6             ...
7       const getNearPlaces = async () => {
```

```
8         const longitude = await AsyncStorage.getItem('longitude')
9         const latitude = await AsyncStorage.getItem('latitude')
10        setLat(latitude)
11        setLon(longitude)
12          try {
13              const requests = tags.map(tag =>
14                  axios.get(
15                      `https://maps.googleapis.com/maps/api/place/
                            nearbysearch/json?location=${latitude},${
                            longitude}&radius=${radius}&keyword=${tag}&key=${
                            apiKey}`
16                  ).then(response => {
17                      return response.data.results.map(place => ({
18                          ...place,
19                          tag: tag
20                      }))
21                  })
22              );
23
24              // Wait for all requests to complete
25              const responses = await Promise.all(requests);
26
27              // Extract the results from each response
28              const sites = responses.flat();
29
30              const uniqueSites = Array.from(new Set(sites.map(place =>
                    place.place_id)))
31              .map(placeId => {
32                  return sites.find(place => place.place_id === placeId);
33              });
34
35              setPlaces(uniqueSites);
36
37          } catch (error) {
38              console.error(error);
39          }
40      };
41                      ...
```

Primero se inicializan todos los datos necesarios. La variable 'radius' consiste en el radio de búsqueda de sitios. Su calculo está basado en el tiempo disponible, multiplicado por la velocidad media de caminata de una personada lambda. Se divide en tres, porque, para poder disfrutar del sitio, el tiempo de ida no puede ser superior al tiempo de estancia en el sitio. La latitud y longitud se recuperan desde el asyncStorage y los tags, que corresponden a las aficiones se pasan por navegación. Finalmente, por cada elemento en el array de tags, se hace una petición a la API de Google Maps en la linea 15, con los datos pasados y el 'keyword' siendo el tag. Esta petición devolverá todos los sitios correspondientes al tag en el radio especificado partiendo de la posición del usuario. Para asegurase de que no haya sitios duplicados, se crea un nuevo array cuyos objetos son los sitios del resultados de la búsqueda eliminando a los duplicados. Esta función esta llamada dentro de un 'useEffect' para poder generar todos los sitios antes de renderizar la pagina, ya que estos elementos serán necesarios.

Una vez la lista obtenida, por cada elemento se ha calculado la distancia entre el usuario y el sitio usando la formula de Harvesine para determinar la distancia entre dos puntos en una esfera.

6.3.3. API Node.js

La API tiene como objetivo contestar a las peticiones del front-end con objetos de la base de datos. Algunas peticiones eran muy sencillas, otras requerían un poco más de trabajo para poder ser implementadas correctamente. Primero y ante todo, para poder crear una API funcional y que se adapte a las necesidades del proyecto, se ha usado 'Express'[7]. Express es un framework de creación de servicios web minimalista que permite, de forma rápida crear una API funcional, para el manejo de rutos y peticiones HTTP. Para ello, basta con importar express, crear unas rutas y controladores. El código para crear la API es el siguiente :

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();

// Middleware
app.use(cors());
app.use(bodyParser.json());

// Routes
const userRoutes = require('./routes/userRoutes');
const siteRoutes = require('./routes/siteRoutes');

app.use('/api/users', userRoutes);
app.use('/api/sites', siteRoutes);

// Start the server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

A parte de express que ya hemos explicado, se usa también CORS. CORS es una librería que permite mejorar la seguridad del servidor. Permite restringir el acceso al servidor de aplicaciones fuera del domino del servidor. Al tener un servidor y una aplicación ubicadas en localhost, no tiene tanta importancia, pero si se imagina un caso real o un despliegue en un servidor externo, sería interesante restringir el acceso de esta manera. También permite gestionar los métodos permitidos y las cabeceras de las solicitudes. En cuanto a la librería 'BodyParser', permite manejar el cuerpo de las solicitudes HTTP entrantes y analizar su contenido. Gracias a ello, podemos manejar el contenido y los objetos JSON de las peticiones que llegan al servidor. Se definen en este archivo las rutas principales de acceso, siteRoutes, para todas las peticiones relativas a los sitios, y userRoutes, para todas las peticiones relativos a los usuarios. Y finalmente el arranque del servidor con 'app.listen()'. En los archivos siteRoutes y userRoutes, se definen las rutas especificas de acceso a cada cada método, y en siteController y userController es donde se programan los métodos. En el código siguiente esta un ejemplo :

```
//UserRoutes.js
const express = require('express');
const router = express.Router()
const{registerUser,loginUser} = require('../controllers/userController');
router.post('/register',registerUser);
router.post('/login',loginUser);

//UserController.js
exports.registerUser = (req,res) => {
    const { username, password} = req.body;
                        ...
        res.status(201).json({message: 'User registered successfully'});
```

```
13 }
14 exports.loginUser = (req,res) => {
15     const{username,password} = req.body;
16                 ...
17         res.status(200).json({token})
18 }
```

En este ejemplo se definen en userController.js las funciones 'registerUser' y 'loginUser'. Estas funciones están importadas en el archivo UserRoutes.js, y se asocian a una ruta y un método para poder ser llamadas desde nuestra aplicación.

Las peticiones se hacían, la mayoría del tiempo, para obtener objetos relativos a usuarios, como sus aficiones, sus sitios favoritos o su historial. Por lo tanto hace falta identificar rápidamente sin pedir que se conecte en cada intento para mantener una experiencia de usuario agradable. Aquí entran los tokens que se mencionaron. Gracias a la librería 'jsonwebtoken', el servidor puede generar, verificar y decodificar JSON WEB Tokens, o JWT. Estos token permiten la autenticación y autorización de acceso a datos, y el mantenimiento de sesión sin la necesidad de almacenarlo en el estado del servidor. En el momento de la conexión, se genera un token mediante el nombre del usuario y una clave secreta generada previamente y almacenada localmente en el servidor. Ahora, en vez de mandar los datos del usuario, bastará con mandar el token generado. El servidor será capaz de descodificarlo e identificar el usuario. Y para la identificación del usuario, se usa otra librería, 'bcryptjs'. Por motivos de seguridad, las contraseñas no se almacenan en base de datos. Sino, se genera un hash a partir de la contraseña, y es este hash que se almacena en base datos. Cuando un usuario pide conectarse, se recupera el hash almacenado para el usuario pidiendo autenticarse, y se compara con el hash de la contraseña introducida. bcryptjs permite cifrar contraseñas, gracias a una función de hashing resistente a ataques de fuerza bruta. También propone métodos para la comparación de hash. Estas dos librerías se pueden ver en la función que login, que permite abrir una nueva sesión en la aplicación :

```
1  const secret = process.env.JWT_SECRET
2
3  exports.loginUser = (req,res) => {
4      const{username,password} = req.body;
5      const query = 'SELECT * FROM users WHERE username = ?';
6      connection.query(query, [username],(err,results) => {
7          if(err) return res.status(500).json({error: 'Database error'});
8          if(results.length === 0) return res.status(404).json({error: 'User
               not found'})
9          const user = results[0];
10         const passwordIsValid = bcrypt.compareSync(password,user.
               password_hash);
11
12         if(!passwordIsValid) return res.status(401).json({error: 'Invalid
               password'})
13
14         const token = jwt.sign({username:user.username},secret,{expiresIn:'
               24h'})
15
16         res.status(200).json({token})
17     })
18 }
```

Gracias a bodyparser, se puede recuperar los datos de la petición con 'req.body'. Se inicializa una petición a la base de datos para obtener la fila de la tabla 'users' cuyo nombre es el nombre introducido. De esta manera, se recupera el hash de la contraseña. El resultado esta almacenado en la variable 'user' en la linea 9. La comparación de los hash se hace en la linea 10 con el método compareSync de bcrypt. Si los dos hash coinciden, se genera un

token a partir del nombre de usuario, el secreto 'secret', y una validez de 24h. Este token es lo que devuelve la función a la aplicación para que pueda luego hacer peticiones con el token para obtener los datos del usuario.

```
const logIn = () => {
        ...
        axios.post("http://192.168.18.12:5000/api/users/login",userData).
            then(res => {console.log(res.data,res.status)
        if(res.status == 200){
            AsyncStorage.setItem('token',res.data.token);
        ...

    }
```

En este codigo de la pantalla de login, se puede observar como, tras mandar una solicitud a la API con axios, se inicializa un nuevo token con AsynStorage, para poder luego usar en cualquier pagina de la aplicación.

6.3.4. Base de datos MySQL

Para responder a las necesidades de la aplicación se ha creado una base de datos acorde al modelo descrito en la figura 5.2. Mediante scripts sql, es posible iniciar la creación de todas las tablas en una sola ejecución. Además de simplemente crear las tablas, también es posible añadir reglas sobre sus propiedades. Estas reglas permiten ajustar el comportamiento de la base de datos acorde a las necesidades de la aplicaciones y a su funciones. Se muestra a modo de ejemplo el script de creación de la tabla 'user' cuyo objetivo es almacenar todos los usuarios de la aplicación :

```
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL
);
```

Cada usuario se identifica con numero, el id. Este numero no es relevante en la aplicación y su única función es la identificación. Por lo tanto, SQL no das la opción de generar automáticamente este numero con cada nuevo usuario, con la función 'AUTO_INCREMENT'. De esta forma, cada nuevo usuario tendrá como id uno más el id del usuario anterior. Además, el nombre de usuario es único, por la tanto, se añade la regla 'UNIQUE' para no poder tener dos usuarios con el mismo nombre. Finalmente, como un usuario tiene que tener un nombre y una contraseña, estos campos no pueden ser vacíos. Por la tanto, se añade la restricción 'NOT NULL' que impide la creación de un nuevo usuario si dicho campo no tiene valor.

Para las tablas que representan las relaciones entre un sitio y un usuario, era necesario que cada tabla tenga como clave primaria la combinación de las claves primarias del sitio y del usuario. De esta forma, cada usuario puede guardar varios sitios, y cada sitio puede ser guardado por varios usuario. Esto, en SQL, se puede hacer la siguiente forma :

```
CREATE TABLE favorite (
    site_id INT NOT NULL,
    user_id INT NOT NULL,
    PRIMARY KEY (site_id, user_id),
    FOREIGN KEY (site_id) REFERENCES sites(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

De esta manera, la claves ajenas, o 'FOREIGN KEY', permite relacionar los atributos 'site_id' y 'user_id' con su id en su tabla correspondiente. La instrucción 'ON DELETE

CASCADE' permite, si un usuario o un sitio vendría a ser eliminado de la base de datos, eliminar también sus referencias en otras tablas.De la misma manera se creó la tabla historia, que tenía la misma lógica pero una función distinta.

De su lado, el servidor se conecta a la base de datos, y efectúa la consulta en función de las peticiones de la aplicación. La función 'añadirCommentario', dado su nivel de complejidad más alto, es un buen ejemplo para entender el funcionamiento del workflow de la aplicación. El código de dicha función es el siguiente :

```
1  exports.addComment = (req, res) => {
2    const { token, site_name, address, lat, lon, comment } = req.body;
3                    ...
4    const addSiteQuery = `
5      INSERT INTO sites (name, address, latitude, longitude)
6      SELECT ?, ?, ?, ?
7      WHERE NOT EXISTS (
8      SELECT 1 FROM sites WHERE name = ? AND address = ? AND latitude = ?
           AND longitude = ?
9      );
10   `;
11   const commentQuery = `
12     INSERT INTO comments (user_id, site_id, comment)
13     SELECT u.id, s.id, ?
14     FROM users u, sites s
15     WHERE u.username = ? AND s.name = ?
16     LIMIT 1
17     ON DUPLICATE KEY UPDATE comment = VALUES(comment)
18   `;
19   connection.beginTransaction((err) => {
20           ...
21     connection.query(addSiteQuery, [site_name, address, lat, lon,
         site_name, address, lat, lon], (error, results) => {
22       if (error) {
23       return connection.rollback(() => {
24           ...
25       }
26       connection.query(commentQuery, [comment, user, site_name], (error,
           results) => {
27         if (error) {
28         return connection.rollback(() => {
29           ...
30         }
31           ...
32         connection.commit((err) => {
33           if (err) {
34           return connection.rollback(() => {
35           ...
36           }
37           return res.status(200).send('Comment added successfully');
38         });
39       });
40     });
41   });
42 };
```

Una de las preguntas importantes de este trabajo era cunado se tenían que insertar los sitios en base de datos. Por tema de recursos, rendimiento y espacio en la base de datos, no era viable insertar en base de datos cada sitio devuelto por el componente 'listado'. Por lo tanto se tomó la decisión de insertar los sitos en base de datos únicamente cuando había una interacción directa con el usuario y que tener el sitio en base de datos era necesario. Estos caso son : cuando el usuario quiere añadir un sitio a sus favoritos, cuando un usuario visita un sitio, cuando un usuario deja un comentario y cuando un usuario deja una

calificación. Estos cuatro casos, para ser resueltos, necesitan tener en base de datos los sitios, para poder crear una relación entre el sitio y el usuario. Por lo tanto, en todos estos casos, se tiene que insertar el sitio en base de datos si dicho sitio no existe. Esto se hace con la petición 'addSiteQuery' en linea 4. La condición 'WHERE NOT EXISITS (SELECT 1 FROM...) permite asegurarse de que, si el sitio no existe, se añade una nueva entrada en la tabla de sitios. Tras ejecutar esta instrucción, se procede a iniciar la siguiente, 'CommentQuery' que permite añadir un comentario. Es importante notar, en la linea 17, que si ya existe un comentario de este usuario para este sitio, en vez de devolver un error, se modifica el valor del comentario con un 'UPDATE'. Al estar ejecutando dos consultas en una sola función, se pudiera dar el caso en el que solo se ejecute una, y la otra devolviera un error. Esto podría provocar inconsistencia de datos, o más errores. Para evitarlo, se usa el método 'beginTransaction' que permite ejecutar de manera segura varias consultas en una sola transacción atómica. Si alguna viene a fallar, todas las otras operaciones se revierten gracias al 'connection.rollback'. Si ninguna da error, se ejecuta entonces el 'connection.commit' que permite confirman la transacción y guardar definitivamente los cambios.

6.3.5. Despliegue de la solución

Una vez acabado todos los componentes básicos y para comprobar el funcionamiento general de la solución, se ha utilizado la herramienta de código abierto XAMPP. Proporciona un entorno completo de servidor local que permite probar los servicios desarrollados. Permite levantar un servidor MySQL al que nuestra API se podrá conectar. Después de eso se tiene que arrancar la API, con el comando 'node.js index.js', y finalmente la aplicación. Si todo se ha configurado correctamente, se puede empezar a probar la aplicación y sus funcionalidades. Sin embargo falta un detalle importante. Estando trabajando en una aplicación móvil, hace falta un móvil para probar el código. Existen dos soluciones, usar un dispositivo virtual, o conectar un móvil. La primera opción fue la más cómoda durante el inicio del desarrollo, pero dio problemas cuando se necesitaba pedir permiso para recuperar la posición del usuario. Entonces se paso a la segunda opción, usar un aparato físico conectado al ordenador.

CAPÍTULO 7
Producto desarrollado

Tras completar el desarrollo de la solución detallada en el capitulo anterior, se ha obtenido una aplicación perfectamente funcional y utilizable, con todas las funciones previstas inicialmente. Para demostrarlo, se ha tomado capturas de pantalla en todas las situaciones permitidas por la aplicación y detallada en los requisitos y escenarios de la sección 4.2.

7.1 Inicio de sesión

Al abrir la aplicación, nos encontramos con la pagina de login (Figura 7.1), que nos da la opción de conectarse a una cuenta o crear una cuenta.

Figura 7.1: Pagina de conexión

Se crea un nuevo usuario, llamado Juan, cuyas aficiones son el cine, los videojuegos y la música. Tras rellenar los campos de contraseña y de nombre de usuario (Figura 7.2), se abre otra ventana para introducir las aficiones (Figura 7.3).

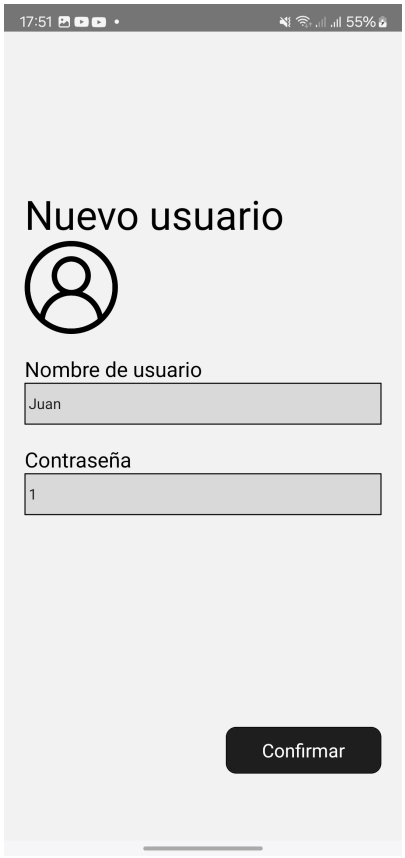

Figura 7.2: Pagina de creación de perfil (1)

Figura 7.3: Pagina de creación de perfil (2)

Una vez creado el perfil, se abre directamente la pagina principal de la aplicación, mostrada en la figura 7.4.

Figura 7.4: Pagina de inicio

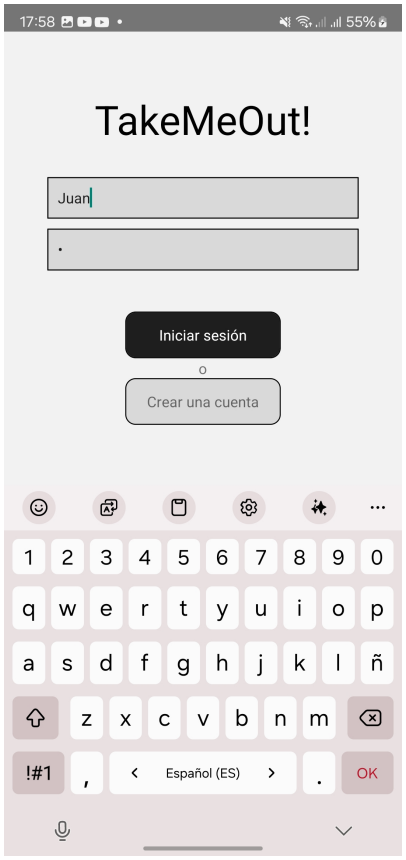

Figura 7.5: Pantalla de Inicio de sesión

Con un nuevo perfil creado, se podrá acceder directamente a la pagina de inicio conectado con los credenciales elegidos (Figura 7.5). Si se quisiera cambiar la contraseña o las aficiones, el usuario podría hacerlo vía el botón con el icono de usuario que aparece en la mayoría de paginas una vez se ha iniciado sesión. Al darle, se abre la pantalla de modificación del perfil que aparece en la figura 7.6. Se permite cambiar de contraseña y modificar la lista de aficiones, añadiendo nuevas, o quitando antiguas.

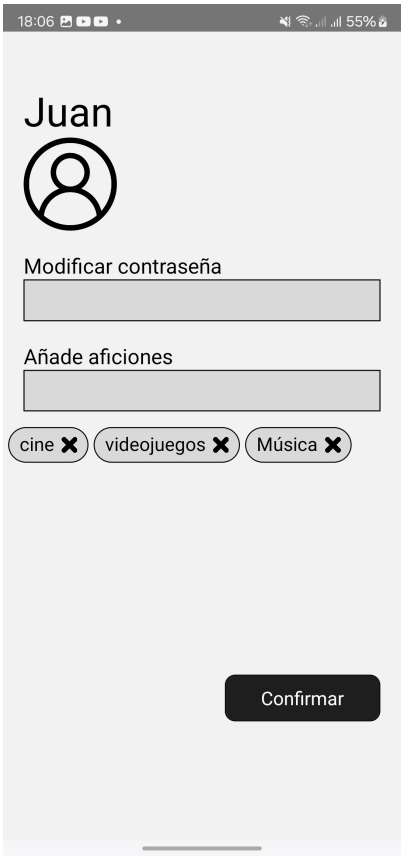

Figura 7.6: Pantalla de modificación de perfil

7.2 Búsqueda de sitios

En la pagina inicial, se pide introducir el tiempo disponible. Con el fin de tener una interfaz más adaptada a las necesidades de los usuarios, el tiempo se incrementa, en minutos, por intervalos de quince. En este caso, introducimos cuarenta y cinco minutos como lo muestra la figura 7.7

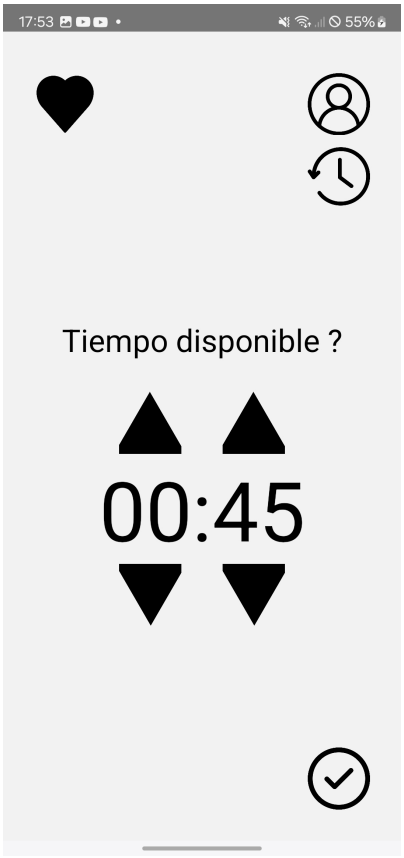

Figura 7.7: Introducción del tiempo disponible

Al darle al botón, se abre una nueva venta, devolviendo un listado de sitios cercanos, con la distancia, una nota, el nombre y sobretodo en cual de las aficiones del usuario dicho sitio encaja. Se muestra ejemplos de esta pagina en las figuras 7.8 y 7.9.

Figura 7.8: Pagina listando los sitios cercanos según los gustos del usuario Juan (1)

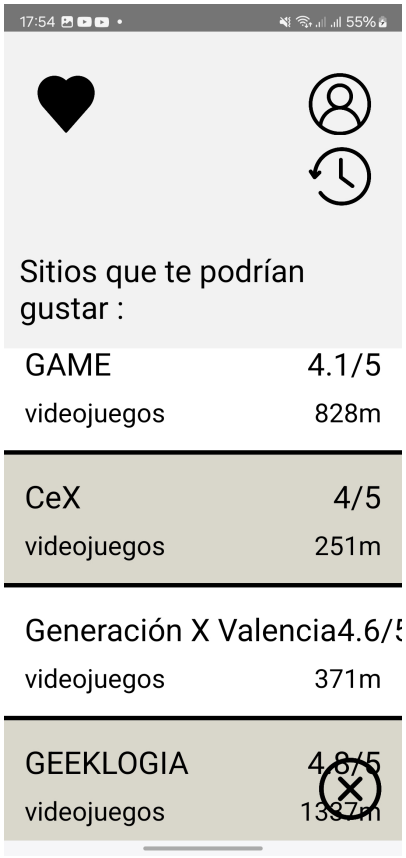

Figura 7.9: Pagina listando los sitios cercanos según los gustos del usuario Juan (2)

Una vez elegido un sitio, se le puede cliquar encima para abrir la pagina de detalles. Aquí se puede ver donde se sitúa el sitio en un mapa, su nota, y los comentarios asociados. En el ejemplo de la figura 7.10, aparece el comentario de otro usuario contado su experiencia. Con el botón 'Iniciar', se puede iniciar un trayecto hacía el sitio. En cuyo caso se abrirá la aplicación de navegación predeterminada, aquí, Google Maps, como lo muestra la figura 7.11.

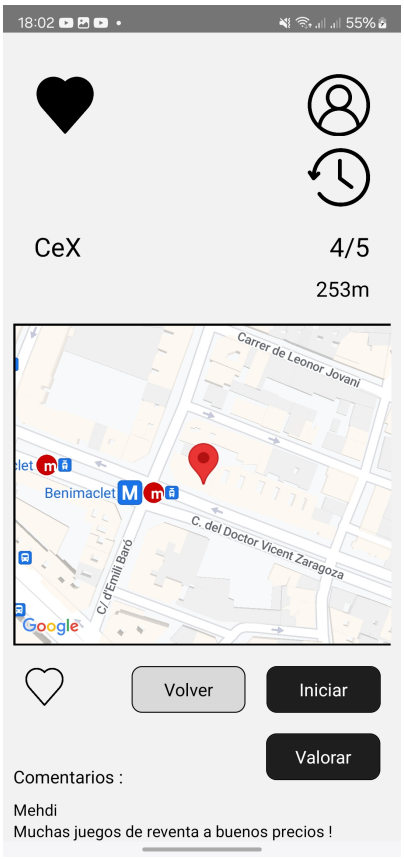

Figura 7.10: Pagina de detalles de la tienda CeX

Figura 7.11: Pagina de Google Maps apuntado al sitio elegido en la aplicación.

7.3 Interacciones con los sitios

Varias interacciones son posibles con los sitios como dejar un comentario, una nota o añadir el sitio a sus sitios favoritos. Se han ejecutado todo estos casos en esta demostración. Primero, tras haber llegado a la tienda, el usuario Juan decide poner una nota y un comentario, gracias al botón 'Valorar' que aparece en la pantalla de detalles de la figura 7.10. Se abre la pantalla de la figura 7.12 que permite dejar una nota y un comentario. Además, Juan decide añadir el sitio a sus favoritos pulsando el botón en forma de corazón. Un vez pulsado, el corazón se rellena, indicando que el sitio se guardo correctamente como favorito, como aparece la figura 7.13

Figura 7.12: Pagina de valoración

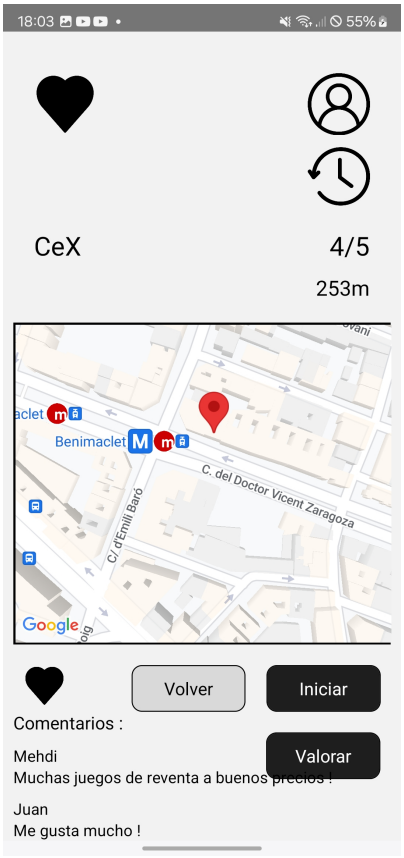

Figura 7.13: Sitio añadido a los favoritos.

Para poder comprobarlo, se puede pulsar el botón en forma de corazón del header. Los botones del header tiene tres funcionalidades, el corazón permite abrir la lista de favoritos, el reloj permite abrir el historial y el icono de usuario permite modificar las informaciones del usuario. La lista de sitios favoritos se muestra en la figura 7.14. El historial de la figura 7.15 funciona de la misma manera y devuelve todos los sitios visitados, es decir, los sitios en los que se ha pulsado el botón 'Iniciar'. Dándole a cualquier sitio de ambas paginas, se abre la pagina de detalles del sitio. Se pueden quitar de estas listos con el botón en forma de 'X'.

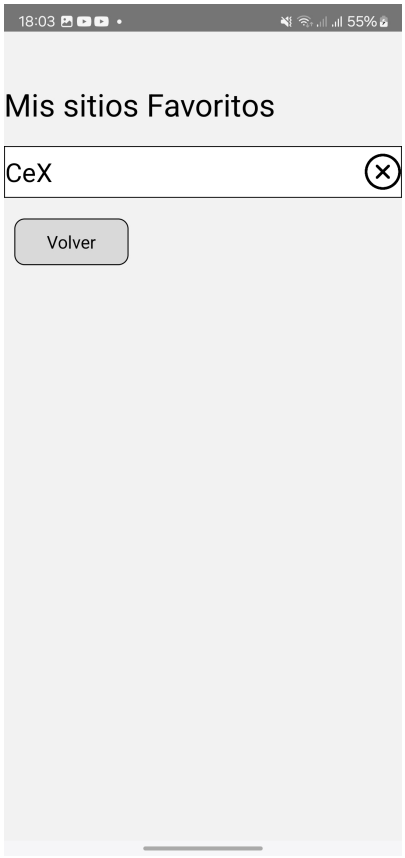

Figura 7.14: Pantalla devolviendo el listado de sitios favoritos.

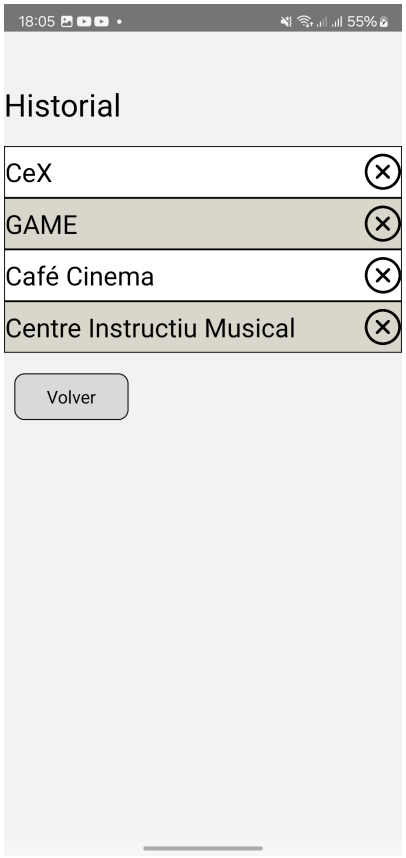

Figura 7.15: Pantalla devolviendo el listado de sitios visitados.

CAPÍTULO 8
Validación

Una vez acabado el producto, es muy importante pasar por un proceso de validación. Este proyecto viene a ser una aplicación que podría ser usada por publico muy amplio. Por lo tanto un único equipo de desarrolladores no puede pensar en todo lo que haría un usuario y prever todos los casos posibles. Además, dado que la aplicación se tiene que usar fuera y estando en movimiento, es complicado ponerla a prueba en una situación real dado que tiene que comunicarse con una API y una base de datos que habría que desplegar en un servidor real. Afortunadamente, existen métodos de validación teóricos que nos permiten, tras un proceso de análisis de la interfaz, determinar sus puntos claves, sus falles y los puntos a mejorar.

8.1 Evaluación heurística

El método de evaluación heurístico[10] permite identificar los fallos de usabilidad en una interfaz basándose en los principios heurísticos de Jakob Nielsen[2]. Es uno de los sistemas más utilizados para la evaluación de interfaces, y tiene como ventaja de no necesitar involucrar a usuarios. Se basa en diez principios que se detallan a continuación, junto con una demostración en el sistema del proyecto o, en el caso en el que fuera ausente, una manera de implementarlo.

- Visibilidad del estado del sistema : En cualquier momento, un usuario tiene que saber que está ocurriendo. En el sistema del proyecto, cada pantalla dispone un titulo. Este titulo tiene como función indicar en que pantalla se encuentra el usuario, como las paginas de sitios o guardados y de historial (Figura 8.1).Estas indicaciones pueden ser de forma interrogativa, como la pagina de inicio cuyo titulo es 'Tiempo disponible?'. El usuario intuye que está en la pagina principal, donde tiene que indicar el tiempo disponible.

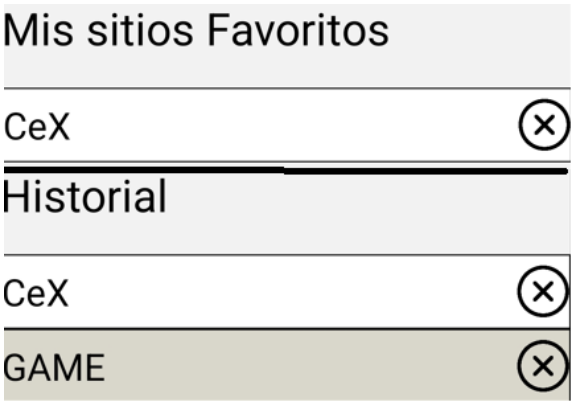

Figura 8.1: Titulo de las pantallas 'Historial' y 'Sitios Favoritos'

- Relación entre el sistema y el mundo real : El sistema debe ser entendible por lo usuarios hablando su idioma. Para ello tiene que emplear un lenguaje natural y conceptos familiares de los usuarios. En casi todo el sistema se utiliza el lenguaje natural para nunca perder al usuario y la mayoría de los botones tiene su función descrita encima. Algunos botones tiene funcionalidades descritas por su icono, como una 'X' para cancelar una operación o quitar algún elemento. Se ha utilizado iconos y símbolos frecuentemente utilizados en otros contextos y otras aplicaciones para que el usuario puede intuir su función solo con verlo. Por lo tanto, se ha utilizado un corazón lleno para abrir le venta de sitios favoritos. La pagina 'Historial' está representada por un reloj con una flecha girando en sentido antihorario, para significar que es algo que ya ha pasado, y el icono de usuario es el mismo icono que se ha utilizado para representar al usuario en la creación del perfil. El botón que permite añadir los sitios a favoritos es un corazón, para significar que esta relacionado con el corazón lleno del header. Una vez pulsado, el corazón se rellena para indicar que este sitio pertenece a la lista de sitios favoritos. Finalmente, lo botones para modificar el tiempo son flechas. Por su orientación y su posición se entiende su funcionamiento. La flechas orientada hacía arriba, encima de los minutos, permite añadir minutos por ejemplo.

- Libertad y control del usuario : Dado que los usuarios se suelen equivocar, el sistema tiene que darle la oportunidad de volver atrás y deshacer lo que se hizo por error. En la aplicación desarrollada, hay muy pocas interacciones que dejen rastro. Por lo tanto,pocas operaciones tienen que reversibles. Concretamente, se da la oportunidad de volver atrás se ha llegado a una pantalla por error, los sitios guardados en las paginas de favoritos y de historial se pueden quitar con solo click, y la lista de aficiones se puede cambiar en el perfil. Sin embargo, no sé ha pensado en el caso en el que un usuario hubiera dejado un comentario por error y quisiera suprimirlo. Por lo tanto, se propone crear una nueva pagina para gestionar las notas y los comentarios.

- Consistencia y estándares : El humano se guía con patrones reconocible en una interfaz. Por lo tanto, es importante mantener los mismos patrones y sus funcionalidades en todo el sistema. Durante todo el desarrollo, se ha mantenido la misma paleta de colores y consistencia entre el diseño de los distintos componentes. Por ejemplo los botones con la función de volver atrás, como lo botones 'Volver' o 'Cancelar' siempre tienen el color y los botones que permiten guardar una operación o iniciar una nueva acción siempre son de color negro.

- Prevención de errores : El diseño de la interfaz debe prevenir la ocurrencia de errores lo máximo posible. Este punto no es tan relevante en este proyecto dado el hecho de que muy pocas operaciones tienen incidencia en el estado del sistema.

- Reconocimiento antes que memoria : El usuario no tiene que memorizar nada. Las instrucciones, objetos, acciones disponibles o opciones tienen que ser visibles y explicitas. En el proyecto, la mayoría de botones y elementos interactivos tienen encima una texto explicando su funcionamiento, como 'Volver', 'iniciar Trayecto' o 'Valorar'. Para el resto de elementos que son iconos, se ha elegido diseños genéricos y elemento reconocibles.

- Flexibilidad y eficiencia de usa : Permitir una experiencia adaptable para todo tipo de usuario. Los usuarios veteranos deberían de tener la opción de acelerar algunos procesos. Sin embargo, la velocidad de ejecución de las tareas depende del servidor y del tiempo de respuesta de la API de google Maps. No del usuario. Por lo tanto, no se ha podido hacer nada.

- Estética y diseño minimalista : Solo se debe incluir información o elementos relevantes. El diseño tiene que ser apurado, elementos sobrecargando la interfaz pueden llegar a crear confusión en los usuarios. Este aspecto se ha tenido en cuenta durante el desarrollo del diseño. Se ha utilizado un numero mínimo de colores para mantener una coherencia entre todas las paginas de la aplicación. Además se ha minimizado el numero de información y de componentes para solo dejar los elementos esenciales.

- Ayudar al usuario a reconocer los errores diagnosticar y recuperarse de los errores : Los mensajes de error tienen que ser claros, dar información sobre el problema y sugerir soluciones. La aplicación proporciona mensajes de error explícitos cuando ocurre un error por causada por el usuario en forma de alerta. Puede ocurrir cuando el usuario introduce demasiadas aficiones, intenta usar un nombre de usuario existente o cuando usa una contraseña incorrecta para conectarse. Se pueden ver ejemplos de dichas en alertas en las figuras 8.2

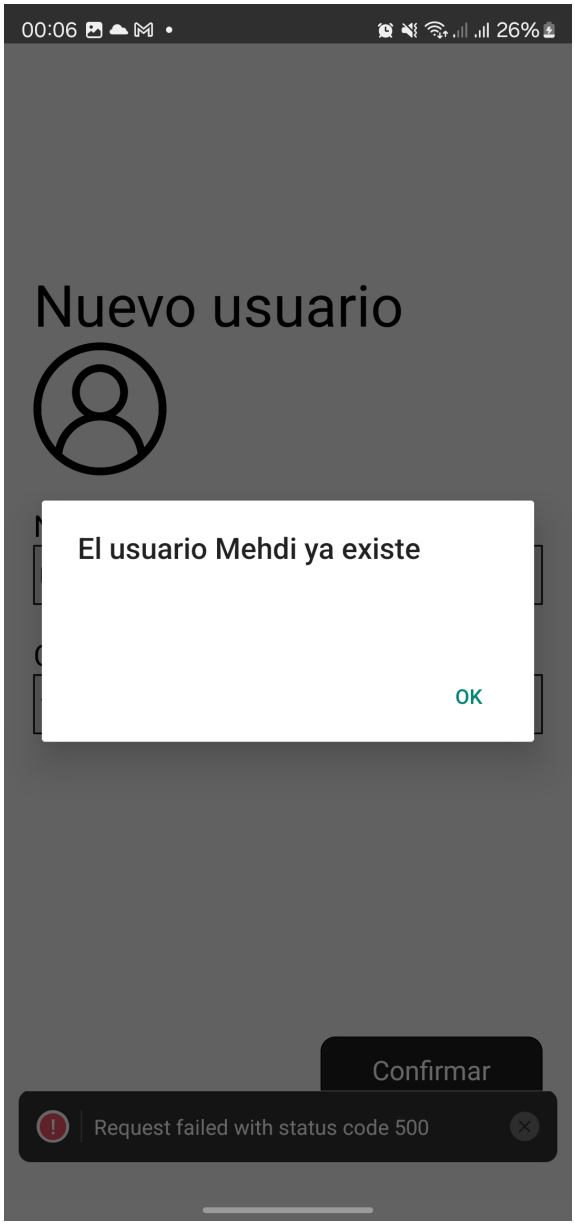

Figura 8.2: Captura de pantalla : Alerta de usuario existente

- Ayuda y documentación : En un caso ideal, la interfaz es tan clara que no se necesita documentación ni ayuda para navegar. Sin embargo, es importante proveer al usuario ayuda y documentación para facilitar su experiencia si se encuentra con algún problema por falta de conocimientos. Dado el tipo de aplicación, no se ha considerado necesario escribir una documentación, ya que toda las funcionalidades son explícitamente descritas por sus elementos y la poca interactividad que tiene el usuario con la aplicación.

8.2 Validación mediante pruebas de usuario

Para confirmar lo que se ha demostrado en la evaluación heurística, se ha un juntado un grupo de nueve personas que representarán un usuario lambda para probar la interfaz. Su tarea no era de probar el funcionamiento de la aplicación, sino más bien su interfaz. Esto tipo de evaluación permite obtener resultados más interesantes que los de una evaluación completamente teórica, porque además de probar la aplicación, se les a

ha pedido rellenar un cuestionario, disponible en el apéndice B. Este tipo de evaluación permite recibir una opinión honesta de personas externas al proyecto y permite cambiar algunos aspectos para pulir la interfaz y mejor algunas funcionalidades antes de pasar a una fase de test más avanzada. En este tipo de pruebas, se les pide a los usuarios realizar una serie de tareas concretas con la aplicación, antes de comentar su experiencia. En este caso, las tareas a realizar fueron las siguiente :

1. Crear una cuenta adaptada a sus preferencias.
2. Introducir un tiempo de una hora y media.
3. Elegir uno de los sitios devueltos por la aplicación.
4. Valorar el sitio dejando una nota y un comentario.
5. Añadir el sitio a su lista de favoritos.
6. Comprobar el contenido de la lista de favoritos.
7. Abrir el historial.
8. Modificar la lista de aficiones del perfil creado quitando una afición y añadiendo otra.

Nueve personas con diferentes perfiles tecnológicos ejecutaron estas acciones en este orden, antes de rellenar el cuestionario. El objetivo fue evaluar la claridad del sistema en las diferentes tareas a realizar. Sus respuestas fueron recompiladas en los siguientes diagramas que aparecen en el apéndice C. Para la mayoría de preguntas, ningún usuario encontró problemas en realizar las tareas. Sin embargo algunos casos fueron más complicados para una pequeña cantidad de personas. Los resultados de dichos casos se detallan a continuación :

Figura 8.3: Resultados de encuesta :Introducir tiempo

El sistema de introducción del tiempo fue problemático para dos usuarios. Puede ser que los botones que se han elegido no fueron lo más adecuado para ellos.

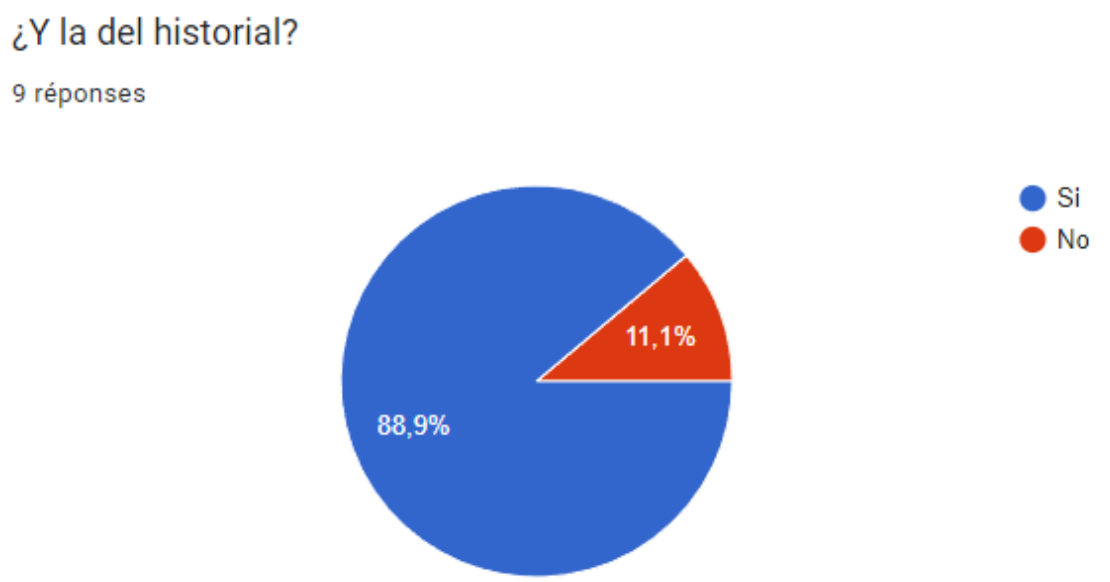

Figura 8.4: Resultados de encuesta : Pantalla del historial

En este caso, un usuario no encontró directamente la pantalla del historial. Al tener perfiles tecnológicos muy variados, puede darse el case de que un usuario no este muy familiarizado con la tecnología, por lo tanto, puede que no relacionó directamente el botón con su función.

Figura 8.5: Resultados de encuesta : Necesidad de más claridad (1)

Según algunos usuarios, algunas pantallas no eran lo suficientemente claras. Dos de tres tuvieron problemas para encontrar el apartado de modificación de las aficiones y un usuario ve necesario añadir más explicación para la introducción del tiempo.

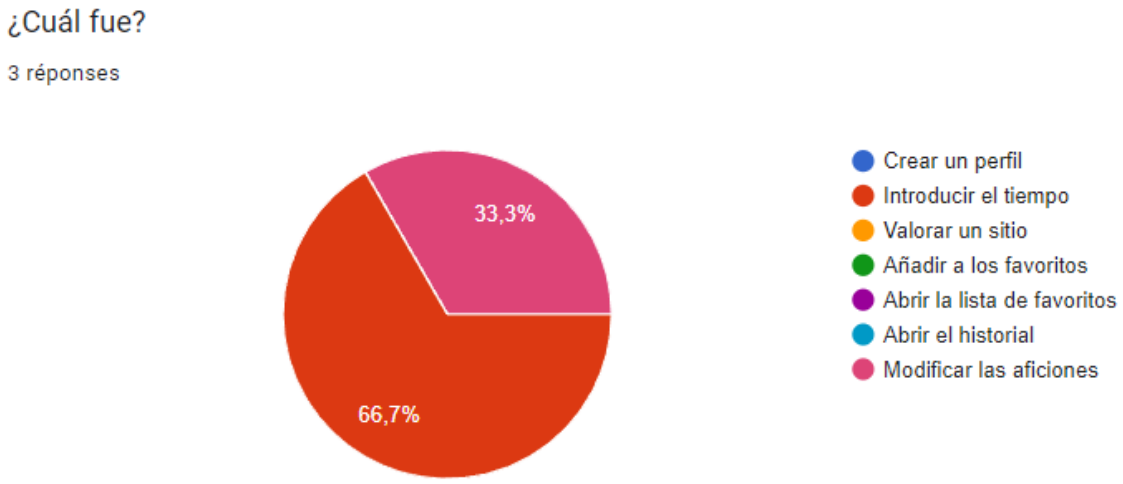

Figura 8.6: Resultados de encuesta : Necesidad de más claridad (2)

CAPÍTULO 9
Conclusiones

En conclusión, este trabajo ha requerido conocimientos adquiridos durante la carrera, ya que se hizo usó de lenguajes de programación y herramientas vistos en muchas asignaturas. En particular para la parte de le base de datos que fue gracias a las distintas asignaturas tratando del tema que se ha podido llevar al cabo. Además de usar tecnologías vistas en distintas asignaturas, los conocimientos adquiridos han hecho mucho más fácil el aprendizaje de nuevas tecnologías como React Native para el desarrollo de aplicaciones móviles. La parte de backend con la creación de la API fue la más relacionada con la rama de tecnología de la información, dado que se trataba del manejo de peticiones HTTP. Por la variedad de temas involucrados en el desarrollo, este trabajo fue la mejor manera de condensar todo los conocimientos, tanto técnicos como teóricos, adquiridos durante los cuatro años del grado de ingeniería informática.

La solución desarrollada cumple finalmente con todos sus objetivos. Permite a un usuario descubrir sitios cerca de su posición, basándose en sus gustos. Se ha pensado en varias posibles maneras de dejar el usuario describir sus gustos. La solución encontrada permite dejar completa libertad al usuario, sin la necesidad de gestionar una lista predeterminada que habría que actualizar. Los usuarios pueden luego compartir opiniones sobre los diferentes sitios lo cual puede ayudar a otros usuarios. Y aunque la interfaz quizá necesita un poco más de trabajo para ser apreciada, la versión del proyecto a la que se llegó es completamente funcional y utilizable si se daría la ocasión de desplegarla.

En cuanto al futuro del proyecto, se pueden imaginar varías maneras de añadir contenido y funcionalidades. Una que se ha pensado durante el desarrollo pero que no se ha conseguido implementar por falta de tiempo es la posibilidad de tener varios tipos de usuarios, uno para los clientes, y otra para los sitios. El gerente de una tienda podría por ejemplo crear un perfil para su comercio y registrarlo directamente en la aplicación. Como tal no parece tener mucha utilidad, ya que solo por estar registrado en Google Maps aparecerá en la aplicación, pero una funcionalidad que podría hacer que valga la pena sería la posibilidad de crear eventos. El gerente de un sitio podría notificar de un evento especial en su tienda, como el estreno de una nueva película para un cine, un concierto para una sala, o cualquier otro tipo de evento. Podría crearse una ventana a parte para ver todos los eventos alrededor en los sitios que corresponden con los gustos del usuario. Y puede hacer que le sea más interesante que simplemente entrar en una tienda o mirar libros en una librería. Esta idea necesita ser pensada mejor para poder ser implementada, y surgió demasiado tarde en el desarrollo. Y sin hablar de nuevas funcionalidades, es interesante ver como personas reales harían uso de la aplicación. Habrán funcionalidades que necesitaran ajustes por el simple hecho no que es imposibles predecir el comportamiento de miles de usuarios frente a una interfaz. Simplemente porque todas las personas son distintas y tuvieron experiencias distintas, algunas funcionalidades pueden parecer obvias por unos, cuando otros serían perdidos. Para evitar este tipo de escenarios se ha-

cen diferentes tipos de evaluaciones como se ha visto en el capitulo anterior, pero nunca es posible tener una interfaz perfecta para todos.

Para acabar, este trabajo ha sido muy interesante, tanto a nivel de desarrollo que de implementación. Dio la oportunidad de adquirido muchos conocimientos nuevos y reforzar los conocimientos existentes, y procuró una gran satisfacción ver por primera vez en pantalla la lista de sitios que se encuentran cerca. Completar este proyecto fue una experiencia muy interesante, tanto a nivel profesional que a nivel personal.

Bibliografía

[1] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Longman Inc, 1999.

[2] Jakob Nielsen. *Usability Engineering*. Academic Press Inc, 1993.

[3] Steve McConnell. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, 1996.

[4] Leonard Richardson, Amundsen Mike, Sam Ruby. *Restful Web APIs: Services for a Changing World*. O'Reilly Media, 2013.

[5] Documentación de React Native, ultima actualización el 29 de agosto del 2023 Consultar en https://reactnative.dev/docs/0.70/environment-setup.

[6] Documentación de React Navigation. Consultar en https://reactnavigation.org.

[7] Documentación de Express, emitido el 08 de octubre del 2022. Consultar en https://expressjs.com.

[8] MySQL Documentation. Consultar en https://dev.mysql.com/doc/.

[9] Tutorial de diagramas de casos de usos, ultima actualización el 26 de junio del 2023 Consultar en https://creately.com/blog/es/diagramas/tutorial-diagrama-caso-de-uso/.

[10] Evaluación heurística, la metodología más utilizada en UX para medir la usabilidad de una interfaz, emitido el 19 de febrero del 2018 Consultar en https://www.mtp.es/blog/experiencia-de-usuario-blog/evaluacion-heuristica-la-usabilidad-una-interfaz/.

APÉNDICE A
Objetivos de Desarrollo Sostenible

El 25 de septiembre de 2015, los líderes mundiales adoptaron un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible. Cada objetivo tiene metas específicas que deben alcanzarse en los próximos 15 años. Una oportunidad para que los países y sus sociedades emprendan un nuevo camino con el que mejorar la vida de todos, sin dejar a nadie atrás. La Agenda define un total de 17 Objetivos de Desarrollo Sostenible (ODS) de aplicación universal para impulsar el crecimiento económico, el compromiso con las necesidades sociales y la protección del medio ambiente. A continuación se presentan los ODS :

1. Fin de la pobreza.
2. Hambre cero.
3. Salud y bienestar.
4. Educación de calidad.
5. Igualdad de género.
6. Agua limpia y saneamiento.
7. Energía asequible y no contaminante.
8. Trabajo decente y crecimiento económico.
9. Industria, innovación e infraestructuras.
10. Reducción de las desigualdades.
11. Ciudades y comunidades sostenibles.
12. Producción y consumo responsables.
13. Acción por el clima.
14. Vida submarina.
15. Vida de ecosistemas terrestres.
16. Paz, justicia e instituciones sólidas.
17. Alianzas para lograr objetivos.

ODS	Alto	Medio	Bajo	No Procede
Fin de la pobreza				X
Hambre cero				X
Salud y bienestar		X		
Educación de calidad			X	
Igualdad de género				X
Agua limpia y saneamiento				X
Energía asequible y no contaminante				X
Trabajo decente y crecimiento económico	X			
Industria, innovación e infraestructuras				X
Reducción de las desigualdades			X	
Ciudades y comunidades sostenibles			X	
Producción y consumo responsables		X		
Acción por el clima				X
Vida submarina				X
Vida de ecosistemas terrestres.				X
Paz, justicia e instituciones sólidas				X
Alianzas para lograr objetivos				X

Las aplicaciones móviles que recomiendan sitios cercanos basados en los gustos del usuario representan una intersección interesante entre la tecnología, los datos y la vida cotidiana. Este tipo de herramientas no solo facilitan la vida de las personas al proporcionarles información sobre su entorno, sino que también promueven la exploración y el descubrimiento de sus barrios y comunidades. A primera vista, podría parecer que su impacto en los Objetivos de Desarrollo Sostenible (ODS) es limitado. Sin embargo, una reflexión más profunda revela que, bajo ciertas condiciones, estas aplicaciones pueden contribuir de manera significativa a varios de los ODS. A continuación, se explora cómo esta tecnología puede relacionarse con los ODS.

- Salud y Bienestar : La aplicación permite que los usuarios descubran lugares para hacer ejercicio al aire libre, como parques, áreas de recreación, senderos y gimnasios. También puede dirigir a los usuarios hacia opciones alimentarias saludables, como restaurantes vegetarianos o tiendas de alimentos orgánicos, contribuyendo así a una mejor nutrición. Sin embargo la aplicación no promueve estos sitios y actividades, es necesario que el usuario defina primero el deporte, los gimnasios y la comida saludable como intereses suyos. Por lo cual solo se relaciona medianamente.

- Educación de calidad : Por haber introducido sitios de educación como museos, librerías o bibliotecas como centro de interés, la aplicación puede devolver un listado interesante de sitios para pasar un rato y poder documentarse y aprender. Su relación es solo baja porque el usuario tiene que estar previamente interesado en estas actividades para que la aplicación pueda devolver sitios del estilo. Además no se hace responsable de la calidad de los sitios, y solo dependerá de las notas y comentarios de otros usuarios saber si un sitio es bueno, o malo.

- Trabajo decente y crecimiento económico : Este tipo de aplicaciones pueden contribuir al crecimiento económico de tiendas pequeñas tienda local e incentivar el consumo local. Promover sitios cercanos como restaurantes, tiendas o cafeterías puede beneficiar a los negocios locales que suelen tener problemas de visibilidad. Estas empresas son fundamentales para la economía local. Al aumentar la visibilidad de estos negocios, la aplicación puede estimular el crecimiento económico dentro de

la comunidad. La promoción del consumo local reduce la dependencia de cadenas de suministro internacionales o de grandes corporaciones. Este tipo de crecimiento económico descentralizado y basado en el consumo local puede apoyar los objetivos del ODS 8.

- Reducción de las desigualdades : Relacionando el ODS 8 con el 9, la promoción de tiendas locales permite aumentar su visibilidad frente a grandes empresas o multinacionales en el mismo sector cuya competencia suele ser injusta frente a pequeños comercios que no tienen los mismos medios y fondos para poder desarrollarse correctamente, y cuya falta de visibildad acabá matando.

- Ciudades y comunidades sostenibles : Una de las contribuciones más claras de una aplicación que permite a los usuarios descubrir su barrio está en el ODS 11: Ciudades y comunidades sostenibles. Al promover la exploración local, las personas pueden reducir su dependencia del transporte motorizado, ya que pueden optar por caminar o utilizar medios de transporte más sostenibles, como bicicletas, para moverse por su comunidad. Además, al fomentar el consumo local, la aplicación apoya el desarrollo de comunidades más autosuficientes y sostenibles. Las ciudades sostenibles son aquellas en las que los residentes pueden acceder fácilmente a los bienes y servicios que necesitan en sus propias comunidades, sin tener que desplazarse grandes distancias.

- Producción y consumo responsables : La aplicación también puede estar relacionada con el ODS 12 en la medida en que fomente la conciencia sobre el consumo local y promueva productos sostenibles. Si los usuarios indican estar interesados prácticas sostenibles, como restaurantes que utilicen ingredientes locales y orgánicos o tiendas que vendan productos de comercio justo, la aplicación puede ayudar al apoyo de un consumo más responsables. Esto contribuye directamente a la reducción de la huella de carbono y al uso más eficiente de los recursos, pilares fundamentales del ODS 12.

El resto de objetivos, al no estar directamente, o indirectamente relacionados con los objetivos y funcionalidades del proyecto, se consideran no procedentes en este análisis.

APÉNDICE B
Formulario enviado a los usuarios

1. ¿Te ha resultado sencillo crear una cuenta?
2. ¿El sistema de introducción del tiempo te ha resultado natural?
3. ¿Los resultados te han parecido pertinentes acorde a las aficiones que has registrado?
4. ¿Has tenido dificultades para valorar el sitio?
5. ¿Has podido añadirlo a tus favoritos?
6. ¿Has encontrado la pantalla de favoritos fácilmente?
7. ¿Y la del historial?
8. ¿Has podido modificar tus aficiones?
9. ¿Has visto necesario explicaciones extras en algunas de estas tares?
10. ¿Cual fue?

APÉNDICE C
Resultados de la encuesta

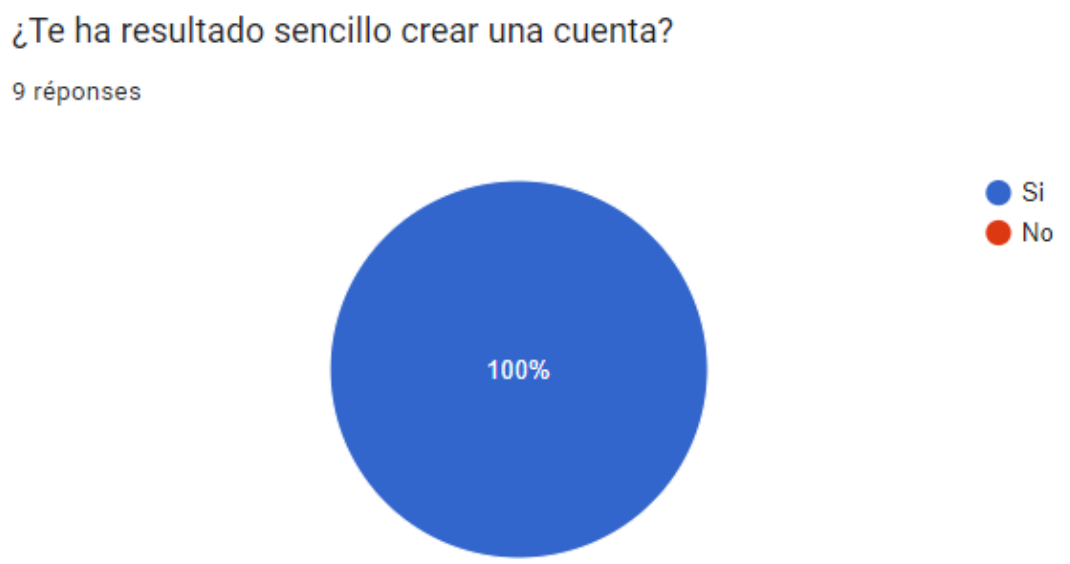

Figura C.1: Resultados de encuesta : Crear cuenta

Figura C.2: Resultados de encuesta :Introducir tiempo

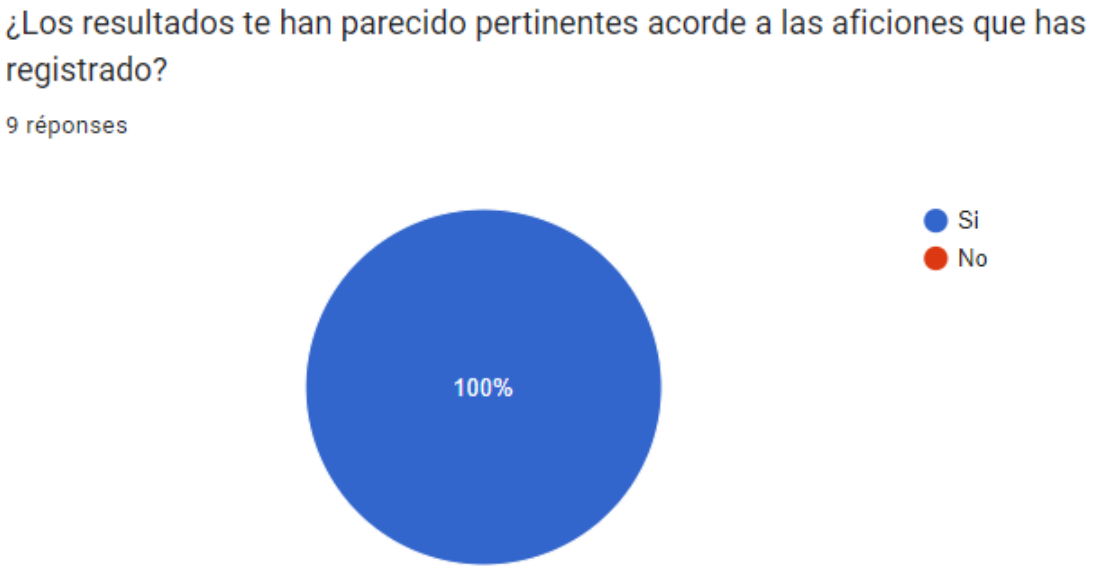

Figura C.3: Resultados de encuesta : Listado de sitios

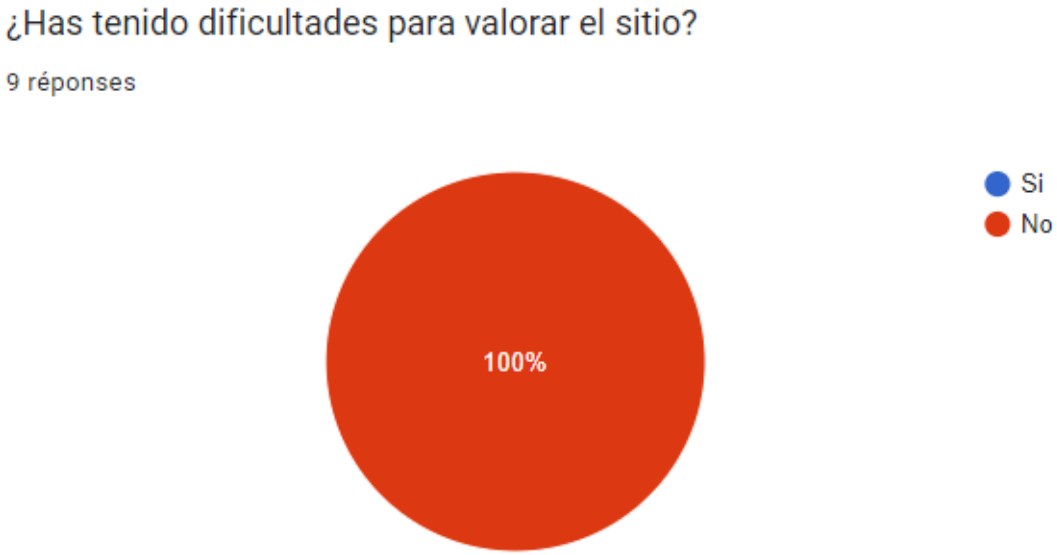

Figura C.4: Resultados de encuesta : Valorar sitios

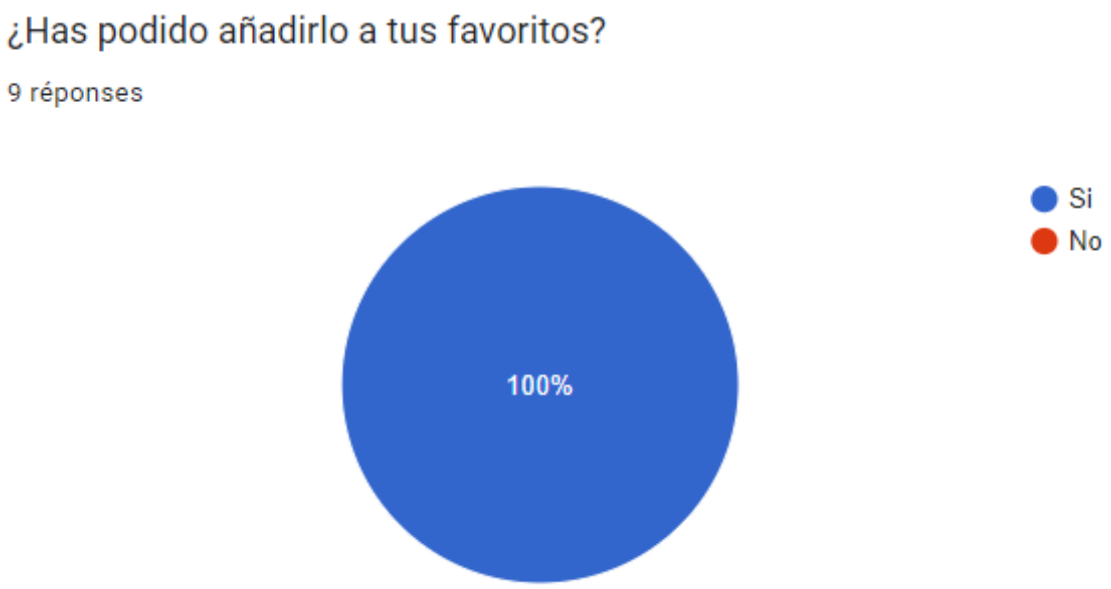

Figura C.5: Resultados de encuesta : añadir a favoritos

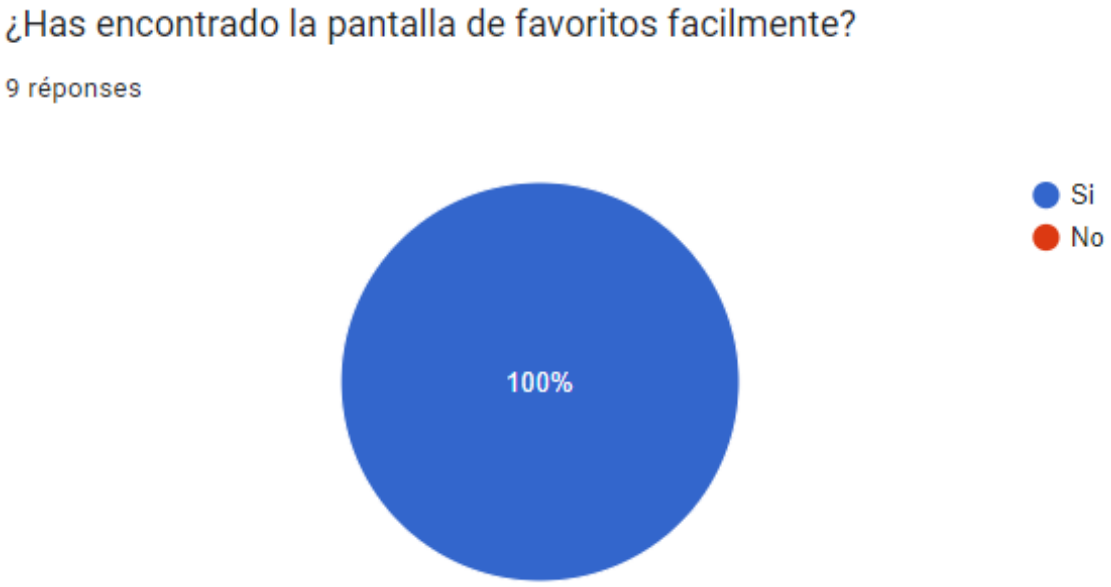

Figura C.6: Resultados de encuesta : Pantalla de favoritos

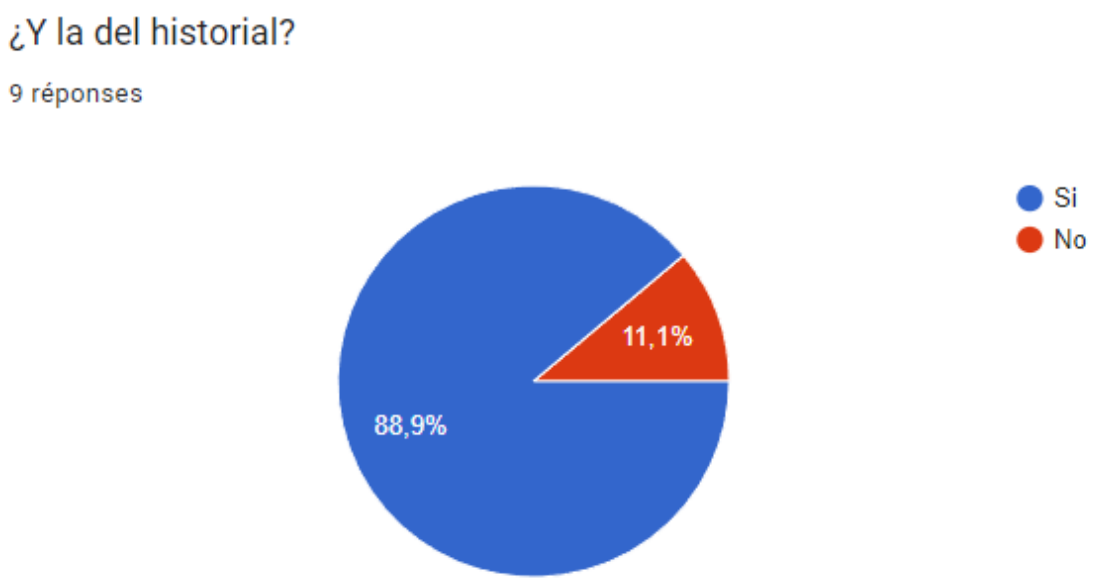

Figura C.7: Resultados de encuesta : Pantalla del historial

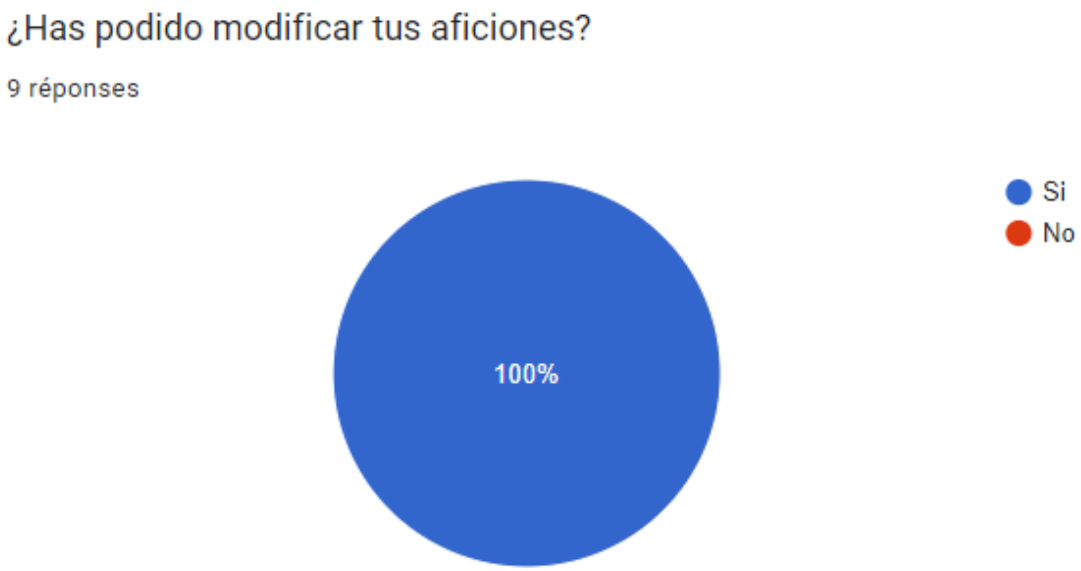

Figura C.8: Resultados de encuesta : Modificar aficiones

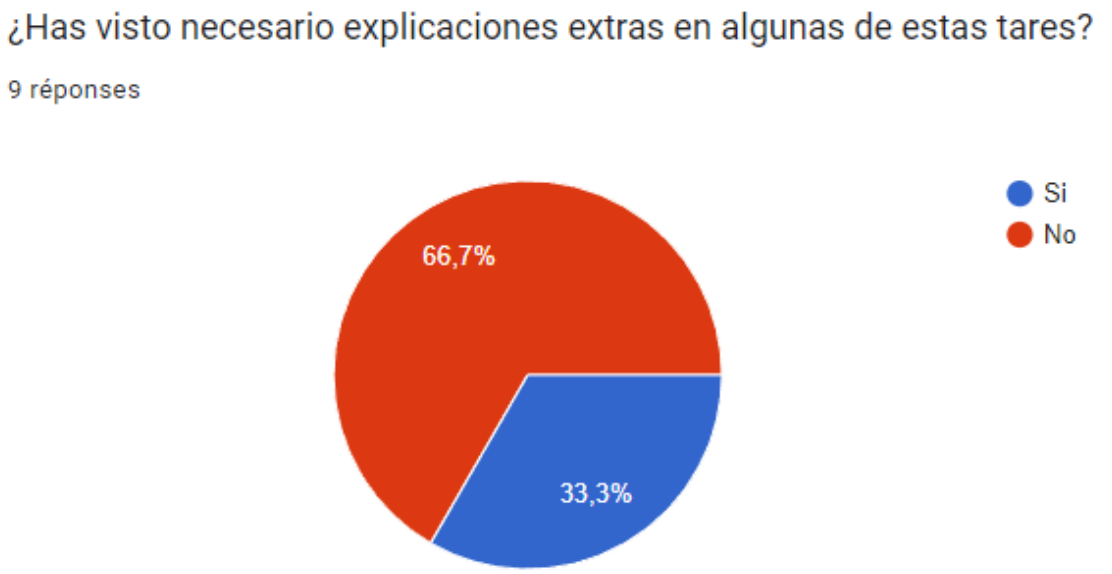

Figura C.9: Resultados de encuesta : Necesidad de más claridad (1)

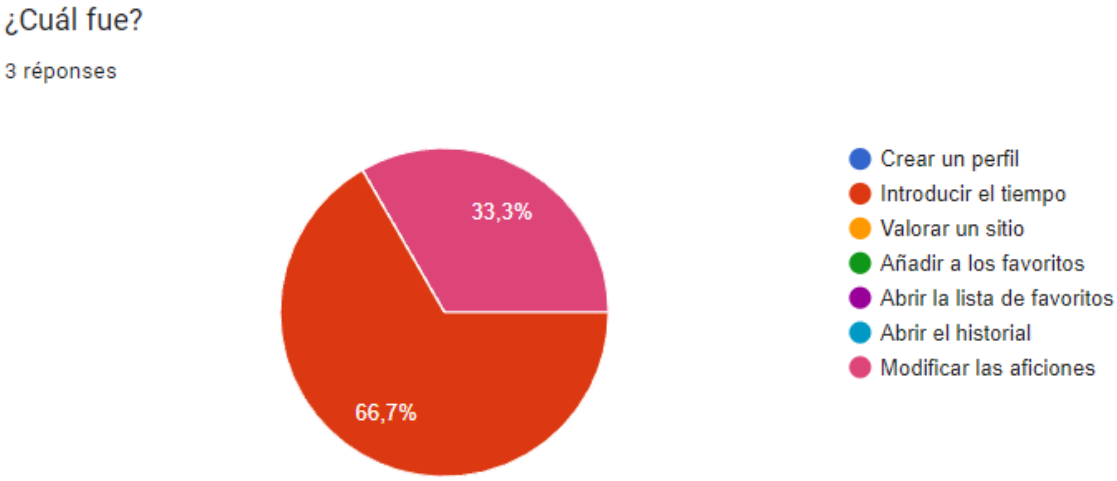

Figura C.10: Resultados de encuesta : Necesidad de más claridad (2)